Security on Internet*

Increasing the resistance of web applications against DDoS attacks

Faculty of Informatics and Information Technologies
Slovak University of Technology
Slovakia, 841 04 Bratislava IV
Email:

***Abstract*—Internet security is a critical issue in the modern world, where the Internet has become an essential part of the daily life of individuals and organizations. One of the most serious cyber threats facing the Internet today is a DDoS (*Distributed Denial-of-Service*) attack, which can cause application overload and subsequent inaccessibility for legitimate users or even financial losses. Given the serious impact of DDoS attacks, it is essential to implement measures to mitigate their impact and ensure the continuity of critical online services. Frequent outages causing the website inaccessible, could result in the organization's services becoming less popular.**

I. Introduction

In recent years, DDoS attacks have become increasingly sophisticated, causing significant damage to web applications and their users. DDoS attacks target multiple RM OSI layers, but the network and application layers are the most significant. Application layer DDoS attacks are particularly challenging to defend against due to the diverse range of attack scenarios.

In section II, we begin by introducing the DDoS attacks. This paper presents an in-depth analysis of application layer DDoS attacks. It discusses a taxonomy of these attacks to better understand their nature in section III. Section IV examines various detection and defense mechanisms currently available for this type of cyber security area. The main goal of this work is to address the challenges posed by application layer DDoS attacks. We propose a novel solution for reactive mitigation that utilizes NGINX access log analysis and dynamic NGINX reconfiguration using a supervisor application. We introduce requirements for this solution in section V and the corresponding design elements in section VI. In section VII, we discuss the implementation process, followed by an evaluation and results of the performed tests against a range of simulated attacks in section VIII. Finally, section IX concludes the results of this work.

Overall, this paper adds to the knowledge of DDoS attacks and offers helpful tips for preventing them, concentrating on application layer attacks.

II. DDoS Attacks

The difference between a DoS (*Denial-of-Service*) attack and a DDoS attack is primarily based on the number of attackers involved. Typically, a DoS attack involves a small number of attackers, and sometimes it can even be an attack by an individual. In contrast, DDoS attacks are characterized by a huge scale in which hundreds or thousands of attackers can participate. It is necessary to emphasize that the attackers are often not human beings, and several bots are involved in the attack [1].

A. Network Layer DDoS Attacks

The oldest type of DDoS attack is known as a network layer DDoS attack. The main goal of this attack is to target the server's network bandwidth. They are volumetric-type attacks and need many network layer packets to limit the bandwidth. However, as time passed, network infrastructure became more reliable, and network layer attack defenses improved. DDoS attacks have recently begun focusing on the application layer. These attacks can be conducted with a relatively modest attack volume compared to network layer assaults. Additionally, they use valid application layer queries, making identifying them challenging for current detection techniques. These attacks can take down a server considerably more quickly and target many resources at the application layer [1].

B. Application Layer DDoS Attacks

These attacks attempt to deplete server resources like CPU, database queries, memory, or socket connections rather than attempting to limit the network bandwidth. Application layer DDoS attacks have some distinguishing characteristics [1]:

- *Legitimate requests:* Ongoing with valid HTTP packets. An attack request and a regular request are essentially identical. The primary distinction is not in content but in intent, and because of this, many packet filters at the network level and even some firewalls at the application layer are unsuccessful in identifying this type of attack [16],
- *Low Traffic Volume:* Most DDoS detection systems now rely on high traffic volume to identify an attack. When an application layer DDoS attack occurs, that strategy fails, rendering the majority of the DDoS detection techniques in use useless,
- *Targeted Strikes:* The other resources will not be harmed by an attack that tries to deplete one resource, but the system will not be able to work as a whole. It is improbable that a protection mechanism for one resource will work for another resource,

*Lecturer: Ing. Rudolf Grežo

- *Resemblance to Flash Crowds:* Because both occurrences are connected to increased HTTP requests to the site, an application layer DDoS attack and a flash crowd are sometimes mistaken. Any defense mechanism must distinguish between an attack and a flash crowd correctly.

Singh et al. perfectly defined the differences between network and application layer DDoS attacks - network layer attacks are prevented at the access zone, which is made up of a variety of network and transport layer hardware, including a router, firewall, IPS (*Intrusion Prevention System*), and other. On the other hand, due to their key characteristics and behavior, application layer DDoS attacks, such as HTTP-GET flood attacks, can penetrate security mechanisms provided by the network and transport layer depicted in the diagram in Figure 1. Security businesses offer WAFs (*Web Application Firewalls*) to defend against these threats. Application layer DDoS attack traffic can easily reach the target server without WAFs [4].

With the rapid pace of IT development, attackers' techniques are also advancing and becoming more sophisticated. Therefore the claim about WAFs is not entirely true, but the description by which they divided these two groups of attacks highlights their key differences.

III. Taxonomy of Application Layer DDoS Attacks

It is crucial to group Application Layer DDoS attacks according to their traits and behaviors in order to understand the nature of these attacks. Praseed and Thiligam in [1] divided them by several aspects.

A. Payload Delivery

First, we can categorize application layer DDoS attacks based on how the payload is delivered - direct and indirect.

1) **Direct Attacks** - attacks are referred to as direct attacks when the attacker sends the attack requests directly to the target server,
2) **Reflected Attacks** - an attacker can also perform an indirect attack, requesting many data from a server while faking the victim's address for application layer protocols that do not require a connection, such as DNS. The server receives the request and transmits the response to the victim directly, resulting in a flood of responses on the victim server [18].

In the case of a direct attack, we can further categorize these direct attacks based on the number of requests or the nature of the payload into symmetric and asymmetric attacks.

1) **Symmetric attacks** - is the most common DDoS attack at the application layer. Like network layer DDoS attacks, many requests are sent to the target web server. However, attacks at the application layer do not require bandwidth throttling to result in a denial of service situation. As a result, the server's resources are now the bottleneck in this scenario and run out long before the server's bandwidth is throttled [16],
2) **Asymmetric Attacks** - sending requests with a high workload to the web server will cause it to crash sooner. Using fewer requests while reducing traffic volume helps the attacker avoid discovery. There are regular size restrictions on the request that a server can accept. To get around this restriction, attackers can use several connections to transmit packets that barely meet the packet size constraint.

B. Protocol Features

Application layer DDoS attacks have primarily employed four different protocols: HTTP, SOAP, DNS, and SIP. Of these, attackers are mostly interested in HTTP and SOAP.

1) HTTP Protocol:

1) **Request fragmentation** - HTTP message can be split into numerous packets. An attacker who fragments his HTTP requests into incredibly small packets can maintain the connection open indefinitely. *Slowloris* is a type of attack that keeps the connections open by providing time-delayed HTTP referers headers and slow, unfinished HTTP requests to the victim web server [3],
2) **Connection refresh** - the HTTP server and any intermediary caches are informed by the HTTP PRAGMA header field that the user wants a new copy of the requested resource. The connection's timeout is reset each time an HTTP PRAGMA is sent. The PRAGMA header can tie up socket resources by keeping connections open indefinitely [1].

2) SOAP Protocol:

1) **Usage of security header** - SOAP makes it possible to encrypt different message parts using various keys. The security header, encrypted with the recipient's public key, must contain all the keys. Large security headers can bring down the system. The server runs out of memory while processing a few malicious packets if the security header is extensively long,
2) **Entity reference** - links that lead to external entities can be included in XML documents, which opens vulnerability for attackers. Referencing large documents in XML messages using an External Entity Reference can exhaust server resources. Such documents are retrieved when the message is being processed. Another example is the Billion Laughs Attack, which creates a memory bomb by defining nested entities within an XML document and making numerous references to other XML elements.

IV. Detection and Defense Mechanisms

Due to their modest traffic volume and use of genuine requests, application-layer DDoS attacks can be difficult to detect. Except for SOAP-based attacks, it is nearly impossible to identify an attack by looking at a single request. To successfully detect an attack at the application layer, one must model and study the interactions among many requests [1].

A. Tools for Application Layer DDoS Attack Detection and Defense

Because this type of attack can only be discovered by closely examining requests and spotting patterns in the request flow, specialized application layer firewalls, IDS (*Intrusion*

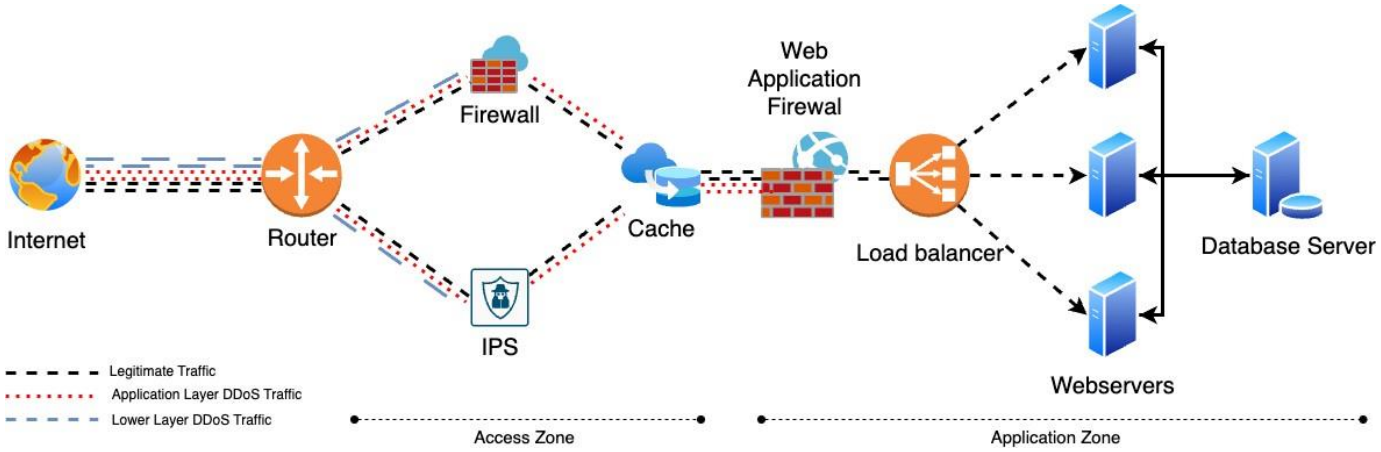

Figure 1: Server Traffic Flow

Detection System), IPS (*Intrusion Prevention System*) and other are required to detect these attacks. While the current security techniques are effective against HTTP floods and other common application layer DDoS attacks, they rely on predefined rules and do not consider typical user behavior when identifying attacks. Asymmetrical attacks are, therefore, likely to slip past these barriers.

DDoS detection systems must function effectively and be able to recognize and stop attacks in real-time. The features listed below are essential for a DDoS detection system:

- *Scalability:* As the number of users increases, so does the stress placed on the detection system. The detection method turns into a bottleneck that the attackers can take advantage of if it cannot expand to accommodate the rising demand to manage a huge number of users without collapsing under the attack,
- *Speed of Detection:* IDS have to work in real-time and analyze all incoming traffic to prevent a delay,
- *Low computational overhead:* The detection mechanism must use the fewest amount of computer resources possible in order not to steal resources from the web application itself,
- *Detection Accuracy:* The detection must be accurate to eliminate false positives and negatives. All legitimate users must be able to access the web application; in contrast, all attackers must be detected and eliminated.

B. *Defense strategies*

1) Using User Puzzles to Block DDoS Attacks: Limiting the number of automated requests may be accomplished easily using user puzzles. A challenge that is simple for a human user to perform but extremely challenging for an automated system. AYAHs (*Are You A Human*) and CAPTCHAs (*Completely Automated Public Turing Test to Tell Computers and Humans Apart*) are mentioned as examples [1]. However, as some other authors noted in their works, user puzzles may prevent DDoS attacks. However, they may also irritate users, drastically diminish their website-visiting experience, and cause further service delays [2]. Another idea is that it is preferable to only serve challenges to users with suspicious behavior detected.

2) Analyzing Request Stream Dynamics: This technique refers to attributes like request quantity and kind, request rate, source IP distribution, etc. This layer of observation concentrates on the minute particulars of a request stream and ignores how a user interacts with a web application and its pages. These techniques are widely used to stop HTTP flooding attacks that rely on huge amounts of HTTP requests. The main sentinels of this approach are:

1) **Estimation of the Traffic** - statistical models and machine learning algorithms are used to estimate the expected traffic using past data. If there is a noticeable deviation from the predicted traffic value, this could be a sign of an attack. The source IP distribution provides more evidence to prove whether the flood is an attack.
2) **Request Statistics** - the number of requests, the request timestamp, the source IP address, header fields, user agents, the number of OK responses, the number of error responses, the number of packets, the number of bytes, the average packet size, the number of packet rates, the rate of bytes, the variance in requests intervals, and the variance in packet sizes from connections. The traffic is probably authentic and accepted if these parameters are within acceptable bounds. These values should be within reasonable bounds; otherwise, the connection should be banned as it is likely malicious. For detecting attacks, the majority of research works combine these features [20].

3) *Analyzing Request Stream Sematics:* The best defensive strategy is to learn typical user behavior, but false positives occasionally appear even with this approach. These detection techniques capture characteristics that characterize a user's interaction with the application. A typical user is unaware of or unable to control his request rate or session rate. The goal of detection techniques is to identify the underlying significance of the incoming request stream, identifying HTTP asymmetric attacks and HTTP flooding attacks. Request semantics can be further divided into two classes [20]:

1) **Request Composition** - using attributes such as HTTP

request rate, HTTP session rate, server documents visited, and access time. Based on the likelihood that a feature will have the observed value, it assigns a suspicion score for each feature for an incoming connection. Total suspicion scores for each feature create the ultimate suspicion score assigned to individual users,

2) **Request Sequence** - the average transition probability and page popularity are widely utilized as clustering features. Frequently, a flooding attack can be recognized by looking at the characteristics of the request stream. The likelihood that a legitimate user produced the incoming request sequence can be determined by calculating the normality of the observed request sequence [3]. Although WAF can offer excellent protection against HTTP floods, big volume HTTP floods might clog the firewalls and cause a new bottleneck.

4) Preventing Slow DDoS Attack: Slow DDoS attacks are usually detected by using a technique to keep track of the unfinished requests in the system. The main lines of protection against slow DDoS attacks continue to be lowering timeout values and restricting the number of connections. By following these preventive measures and installing the appropriate Apache security modules, configuring appropriate IPTables or IDS rules can effectively decrease the attack's magnitude but cannot prevent it from happening. Unfortunately, these safeguards may cause legitimate users with limited bandwidth abruptly sever their connections [3].

5) Template Matching: Schema hardening and adequate schema validation are common defense strategies against SOAP protocol DDoS attacks. Each Web service has its unique schema that outlines the format of incoming messages. Schema hardening entails putting constraints on the input fields, such as input size and nesting depth limits. In other words, SOAP or XML-based DDoS attacks can be largely avoided by comparing each incoming request against a template.

6) DNS Caches: The most harmful and challenging to detect denial-of-service attacks are those on DNS servers. Identifying whether or not an incoming request is valid is the main challenge in detecting an attack on a DNS server. UDP communications effectively eliminate the ability to identify and block attackers. DNS caches, which store infrastructure records for extended periods, are a way to reduce the impact of attacks on DNS servers. Although it does not stop attacks on DNS servers, this can assist in softening the impact of denial of service attacks on a DNS server because the caching servers can continue to operate in such a scenario [1].

V. Requirements Specification

We outline the specifications for our DDoS protection mechanism solution in this section. Given that HTTP is now the most popular application layer protocol used by web applications, we decided to focus on it based on the thorough research of application layer DDoS attack types we have concluded in the section III. Additionally, as there is no proof of any reflected DDoS attack targeted at HTTP services, we will concentrate our efforts on direct attacks with primarily symmetric payloads. The defense mechanism that is being presented must be able to:

1) use as many existing effective solutions as possible in order to combine multiple defense strategies,
2) parse and export access logs in real-time,
3) monitor incoming traffic to the HTTP server based on the collected logs,
4) detect an ongoing attack and identify an attacker among legitimate users,
5) reactively reconfigure the web server to prevent the attacker from accessing the web application.

VI. Solution Design

In this section, we will define the elementary parts of our solution. We seek to identify attackers and separate genuine traffic from malicious traffic through real-time metric analysis. We intend to apply reactive defense techniques to proactively reconfigure web server under attack and utilize web server hardening to increase security by employing preventive measures. We separate our infrastructure into multiple components to meet the requirements listed in section V.

A. Reverse Web Proxy

An internal network's servers receive requests from the Internet through a reverse proxy. Requesters may not be aware of the internal network when they connect through the proxy. In reverse proxy, many application firewall features can defend web applications against common web-based threats [3]. Our solution will use the NGINX Web server, which provides several useful tools to meet our requirements. To define upstream servers and reverse proxy requests to those servers, it makes use of configuration settings like *proxy pass* or even the potential to utilize it as an effective HTTP load balancer to split traffic across multiple application servers and enhance the speed, scalability, and reliability of web services [5].

Moreover, it offers logging features that may be customized to record desired information about requests and errors. Listing 1 illustrates the typical access log format. Listing 2 shows a log entry that fits this format in detail.

Respective fields in the default log format are defined as follows [5]:

- *$remote_addr* - client IP address,
- *$remote_user* - user name supplied with the Basic authentication,
- *$time_local* - server's local time of the processed request,
- *$request* - full original request,
- *$status* - HTTP status code of the response,
- *$body_bytes_sent* - size of body sent to a client,
- *$http_referer* - value of *Referer* HTTP header,
- *$http_user_agent* - value of *User-Agent* HTTP header.

B. Logging and Monitoring

We can already get some interesting conclusions from the default log entry in Listing 2. Based on his IP address (172.17.0.1), we can recognize the client and know his purpose (HTTP GET for route /). Also, we are aware of the date and time of the request (7. May 2023, 7:21:02), the response status (200), and the size of the answer (615 bytes). In addition, we are aware of the device used to send the request (Mac OS X).

Listing 1 *Default NGINX access log format.*

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';
```

Listing 2 *NGINX access log entry.*

```
172.17.0.1 - - [07/May/2023:07:21:02 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (Macintosh;
↪  Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0
↪  Safari/537.36" "-"
```

We have talked about NGINX and its logging features; in addition to storing logs in log files, NGINX can export logs to a remote syslog server and allows us to modify the standard log format shown in Listing 1. We suggest including the following parameters in the logs [5]:

- *$request_length* - length of the request including request line, header, and request body,
- *$request_time* - request processing duration in seconds with a millisecond resolution.

C. Real-Time Metric Analysis

A crucial component of any security monitoring is log analysis. It can help with identifying malicious actors and their intents. It takes much time to analyze text-based raw logs. This is why logs are often parsed and exported into a time-series database. This type of data storage makes it possible to go through it visually (e.g., various graph representations). Users of the commercial NGINX license have access to a Live Activity Monitoring tool. The tool supports a configurable alerting system based on predefined thresholds, in addition to its visualization techniques [5].

The capacity to track changes is provided by real-time metrics, represented as a series of data points gathered over time. With that being said, employing traditional SQL or NoSQL databases is an insufficient strategy in such a scenario. Time-series databases, like Prometheus or InfluxDB, offer procedures like frequent data sampling and, thus, monitoring to address the problem of real-time data processing. Due to the capabilities of time-series databases, we suggest integrating them into our solution to store metrics, analyze them, and spot anomalies.

D. Attack Detection

In section IV, we have discussed multiple detection and defense strategies against application layer DDoS attacks. For detecting an ongoing DDoS attack on our application, we propose to utilize an analysis of request stream dynamics and slight semantics. We decided to track the following data:

- Number of requests - the total amount of processed HTTP requests.
- Request duration - a summary vector of the total response times in seconds.
- Request size - the total amount of received traffic in bytes.

As previously mentioned, time-series databases enable us to create unique alerts activated when specific criteria are met. We aim to configure an alert to identify suspicious activity in incoming traffic and cherry-pick malicious individuals. A supervisor application should be alerted and perform a reactive mitigation strategy when the threat is recognized. We consider requests from a certain remote address that deviates from all other requests from other IP addresses as evidence of an ongoing DDoS attack based on the analysis we concluded in section III. To assign these anomalies, we will use statistical analysis, aggregating over tracked data over a period of time. The simplest and most effective approach for this work is *z-score*.

$$Z = \frac{x - \mu}{\sigma}$$

As the formula shows, the z-score is simply the raw score (x) minus the mean (μ), divided by the standard deviation (σ). Z-score depicts the number of standard deviations from the mean. In a data set with a normal distribution, a z-score of 0 would indicate that the z-score is the same as the mean, and a z-score of 1 would indicate that it is 1.0 from the mean. 99.7% of the samples should have a z-score between 0 and 3 if the underlying data follows a normal distribution, according to this assumption [8]. In his work [4], Singh mentioned similar detection mechanisms, observing users' access behavior. He presents an opinion that malicious bots behave differently from how regular users would normally act. The detection methods use this distinction to find bots among all users. Each user is given a value or score based on their access behavior, as it is more often known. This number indicates how legitimate a user is in terms of credibility.

E. Attack Mitigation

NGINX is the main entry point for all incoming requests to our infrastructure. As mentioned in subsection VI-A, requests can then be forwarded to the other services that are not exposed to the public Internet. In the previous subsection VI-D, we have discussed the techniques we will use to detect an attack and distinguish attacking IP addresses. When we know the attacker's or attackers' IP address, we can take advantage of the built-in NGINX feature and restrict access to the web server for those IP addresses. The *allow* and *deny* directives inside the stream context or a server block of the NGINX configuration (see, Listing 3) can be used to permit or prohibit access [5].

As the goal of our defense mechanism is to restrict access, using only *deny* directives will keep the access list cleaner and more readable. To achieve better management of the deny listed IP addresses, we propose to isolate them into separate *access.conf* and include it in our main *nginx.conf* as shown in Listing 4. Using this method, any system component may

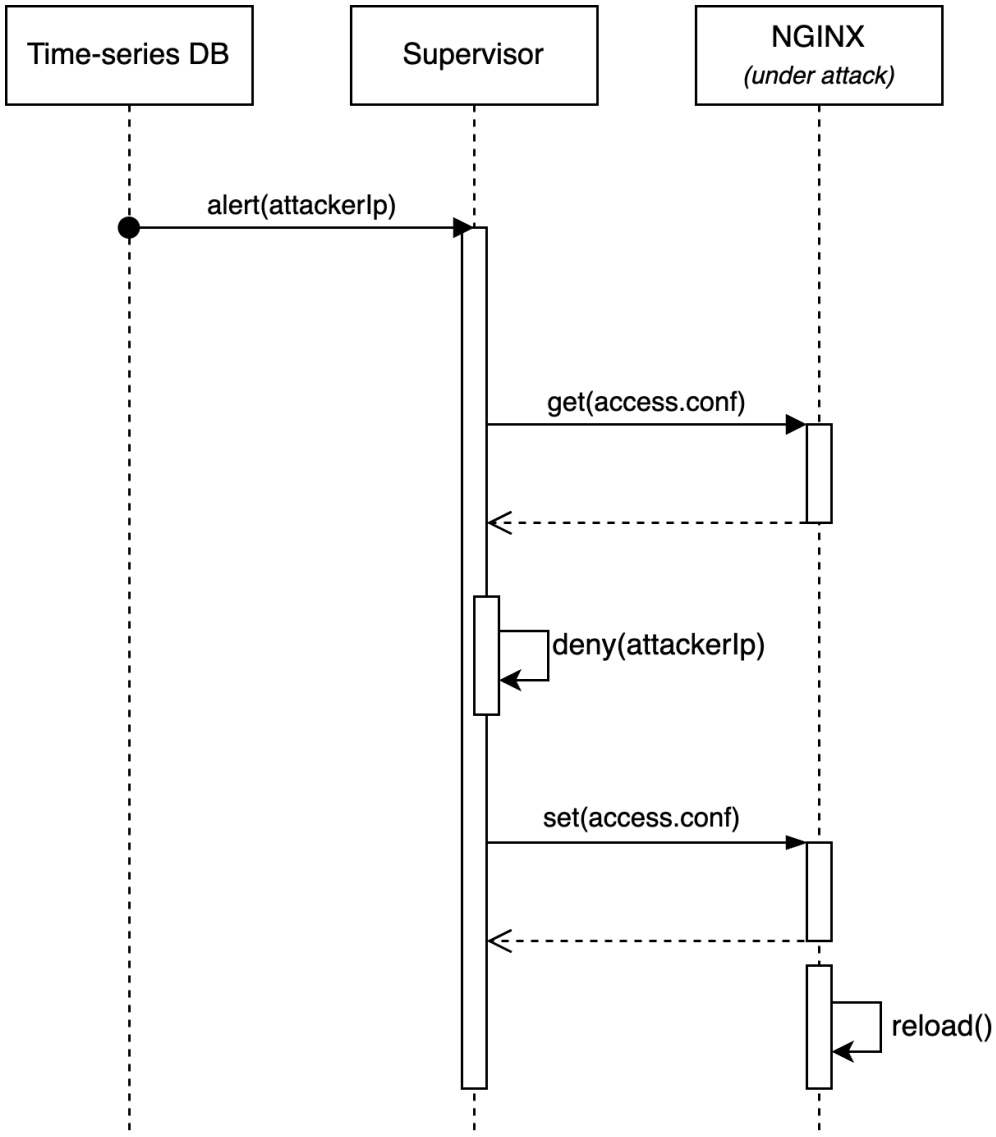

Figure 2: Reactive mitigation using dynamic NGINX reconfiguration

Listing 3 *Default NGINX configuration with access rules.*

```
http {
    #...
    server {
        listen 80;
        listen [::]:80;
        server_name localhost;
        deny   192.168.1.2;
        allow  192.168.1.1/24;
        allow  2001:0db8::/32;
        deny   all;
    }
}
```

instantly retrieve all the information required on the banned IP addresses. For instance, the detection tool can find IP addresses previously denylisted in the global access configuration and exclude them from the calculations for future log analysis.

Listing 4 *Configuration with included access.conf.*

```
http {
    #...
    server {
        listen 80;
        listen [::]:80;
        server_name localhost;
        include /etc/nginx/conf.d/access.conf;
    }
}
```

Using the supervisor application, we suggest achieving this by dynamically reconfiguring the NGINX's *access.conf*. The reconfiguration will be initiated once the monitoring system has detected an attack. Figure 2 depicts the whole reconfiguration process, from the point where the alert gets raised, until the point where reactive reconfiguration strategy is applied, and NGINX is reloaded and reconfigured.

Listing 5 *Log format and syslog server configuration.*

```
log_format format '$remote_addr - $remote_user [$time_local] '
                  '"$request" $request_length $status '
                  '$body_bytes_sent $request_time '
                  '"$http_referer" "$http_user_agent"';

access_log     syslog:server=nginx-exporter:5514,facility=local7,tag=nginx,severity=info,nohostname
↪   format;
```

VII. Implementation

In this section, we describe the implementation of the solution designed in section VI.

A. Tools & Technologies

We already discussed the necessary components and parts of the solution. However, we have not chosen specific tools and technologies except for NGINX as our entry point, web server, and reverse proxy.

1) Docker: To ensure smooth functioning, scaling, and simple management of each component of our solution, we decided to run them in a dockerized environment. Docker is an open platform, thanks to which we can bundle and operate an application in a loosely isolated environment known as a container. Due to the isolation and security, we can run numerous containers concurrently on a single host. We can control our infrastructure using Docker in the same manner that we manage our applications. Additionally, due to its portability and lightweight nature, applications and services may be scaled up or down in almost real-time while workloads are managed dynamically [6]. We will also use **docker-compose**, a tool for defining and running multiple containers. With Compose, we use a YAML file to define our services, configure their private or shared volumes, create a network, and ensure working communication among containers.

2) NGINX: NGINX is a tool mentioned earlier in subsection VI-A. We described its basic functionalities and logging capabilities. We will use NGINX as our web server with a few extra features tailored to our needs. The breakdown of our NGINX configuration is also important. We have edited the default configuration by adding additional properties, as proposed in subsection VI-B. These additions are necessary for further parsing the logs in *NGINX-to-Prometheus exporter*, which will be covered in VII-A3. Furthermore, we set the log export destination to *NGINX-to-Prometheus exporter*. Figure 5 contains the configuration lines we have added.

3) NGINX-to-Prometheus log file exporter : We use the NGINX-to-Prometheus log file exporter [9] to convert the raw logs output from the NGINX container into metrics. This application handles three essential functions:

- receiving raw NGINX logs - application exposes *syslog* endpoint to which NGINX sends access logs as we described this feature in subsection VI-B,
- parsing NGINX logs - application parses raw text NGINX access logs into metrics recognizable by Prometheus,
- exposing metrics - application exposes */metrics* endpoint through which Prometheus collects metrics.

4) Prometheus: Prometheus is a time-series database that collects metrics by scraping HTTP endpoints [7]. We used Prometheus based on the analysis summarized in subsection VI-C. In our example, the *NGINX-to-Prometheus exporter*'s public */metrics* endpoint is used by Prometheus to scrape metrics. The configuration file given in listing 6 shows that we set the scrape interval to 15 seconds.

Listing 6 *Prometheus configuration.*

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: "prometheus"
    scrape_interval: 15s
    metrics_path: "/metrics"
    static_configs:
      - targets: ["nginx-exporter:4040"]
```

A metric, denoted by a specific term, is a physical measurement. Also, to distinguish the properties of a certain metric, we utilize labels (such as *remote_addr*, *status*, or *request*). For instance, *nginx_http_response_count_total{remote_addr="192.168.10.231"}* represents the amount of HTTP requests handled by NGINX coming from IP address 192.168.10.231. Below is the full list of metrics that are provided [9]:

- *<namespace>_http_response_count_total* - the total amount of processed HTTP requests/responses,
- *<namespace>_http_response_size_bytes* - the total amount of transferred content in bytes,
- *<namespace>_http_request_size_bytes* - the total amount of received traffic in bytes,
- *<namespace>_http_upstream_time_seconds* - a summary vector of the upstream response times in seconds,
- *<namespace>_http_upstream_time_seconds_hist* - same as <namespace>_http_upstream_time_seconds, but as a histogram vector,
- *<namespace>_http_response_time_seconds* - a summary vector of the total response times in seconds,
- *<namespace>_http_response_time_seconds_hist* - same as <namespace>_http_response_time_seconds, but as a histogram vector.

5) Grafana: To better understand the scraped metrics and visualize them, we chose Grafana [10]. Grafana is a web-based visualization tool that can query Prometheus data. We also used and enhanced a dashboard that shows various charts of the incoming traffic to the NGINX. The graphs that our dashboard employs are shown in the following images.

Figure 3 depicts the z-score of the request sizes by each client IP that requested some content. We have utilized

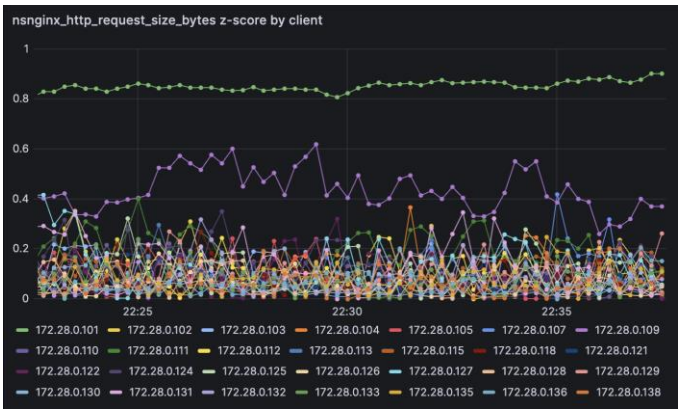

Figure 3: Z-score of request size by client

Prometheus Query Language (PromQL) to query the needed data, enabling us to pick and aggregate time-series data in real-time [7]. After that, Grafana automatically processed and displayed the query results. We improved the visuals by changing the tooltips, legends, and graph styles.

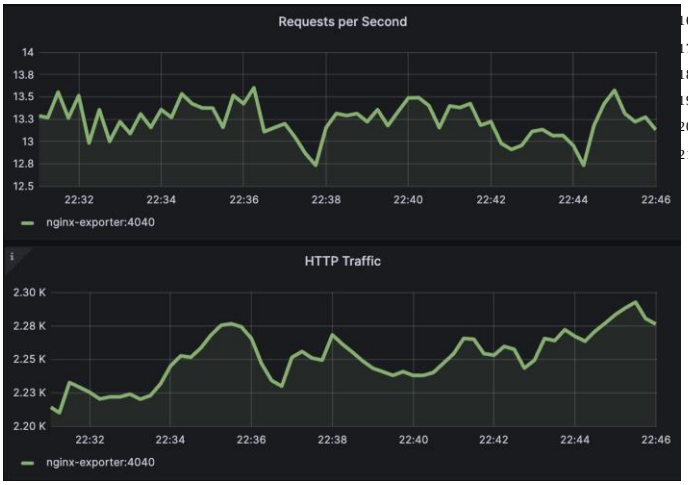

Figure 4: Built-in graphs monitoring traffic

We have used *NGINX Log Metrics [M]* dashboard with a few elementary graphs already created [12]. This dashboard is tailor-made for the NGINX-to-Prometheus exporter and depicts some basic metrics to monitor the flowing traffic. Figure 4 shows two visualizations - Requests per second and HTTP Traffic.

6) Supervisor: The Supervisor application is a simple server-side web application created in *NestJS* framework [11]. The application is implemented with proper logging, and every step of the mitigation process is logged to the standard output, and we can find those logs inside the Supervisor container. We will introduce other functionalities and access module responsible for reactive mitigation in subsubsection VII-B2.

7) Locust : On the Docker network, the client container running *ubuntu:focal* Docker image has a single network interface with several IP addresses (172.28.0.0/16). We created the benchmarking scripts using *Locust* [13], a load-testing tool built with Python. In order to utilize Locust, we created a custom Docker image with *ubuntu:focal* as a base image with a few extra steps like *tty: true*, allowing us to run Ubuntu without exiting it after startup, *cap_add: NET_ADMIN* for managing network interfaces and bash script *provision.sh* (see, Listing 7) for installing necessary *apt* and *Python* libraries and setting up IP addresses for the network interface.

Listing 7 *Bash script provision.sh.*

```
#!/bin/bash

# IP assignment .101-141
for i in {1..41}
do
    if [ $i -lt 10 ]; then
        last_octet="10$i"
    else
        last_octet="1$i"
    fi
    ip addr add 172.28.0.$last_octet/16 dev
    ↪ eth0
done

# Install python & pip
apt-get update
apt-get install -y python3.10
apt-get install -y python3-pip

# Install locust & requests-toolbelt
pip3 install locust
pip install requests-toolbelt
```

We were able to specify end-user behavior on the web using Locust. A BasicUser class's behavior from master thesis [21] for testing content delivery network is shown in listing 8. Besides that, we also defined a similar script for simulating a DDoS attack.

B. Reactive defense

In this subsection, we introduce the implemented extensions we have used to achieve the suggested reactive attack mitigation. The main parts of the reactive defense strategy are described in the following subsubsections. We will cover *Alerting* in subsubsection VII-B1, the *Supervisor Access Module* in VII-B2, and finally principles of dynamic NGINX reconfiguration we achieved in VII-B3.

1) Alerting : The built-in *alertmanager* in Grafana enables customizable alerts to raise when a specific condition is met. In subsection VI-D, we set the attack detection design, and here we create consistent thresholds. We used *PromQL* queries for creating alert conditions. We provide an example of such a query in listing 9 for querying *z-score* of the number of requests for each IP address. Afterward, we drop non-numeric values; each value greater than three fires an alert.

Grafana notifies a webhook HTTP endpoint when a condition persists for at least two minutes. In addition to the metadata, the alert payload also contains the malicious actor's IP address. The HTTP endpoint's implementation, which handles this alert, is covered in the next section.

Listing 8 *Normal user behaviour Locust script.*

```
from locust import HttpUser, task, between
from requests_toolbelt.adapters.source import SourceAddressAdapter
import constants
import random

class BasicUser(HttpUser):
    wait_time = between(1, 5)

    @task(2)
    def get_kek(self):
        self.client.get("/KEKW.png")

    @task(1)
    def get_base(self):
        self.client.get("/")

    @task(30)
    def get_random(self):
        self.client.get("/"+constants.files[
            random.randint(0,
            len(constants.files)-1)
            ])

    def on_start(self):
        client_ip = "172.28.0.1"+"{:02d}".format(random.randrange(0,
        40))
        self.client.mount("http://",
        SourceAddressAdapter(client_ip))
```

2) Supervisor Access Module : We have added an access management module to the Supervisor application responsible for receiving alerts and executing the mitigation process. Our approach is depicted in a sequence diagram in figure 5. We set up an HTTP endpoint through which the Grafana alertmanager can issue alerts when it discovers an attack. Next, Alert Handler starts the mitigation process after raising an internal event ddos.detected. It first retrieves the current blocked IPs from *access.conf*, shared between the NGINX container and Supervisor container via Docker volumes. Secondly, the Access Modifier service checks whether or not the IP is already blocked. Suppose IP is blocked in *access.conf*, the process of mitigation finishes. If IP is not denylisted, the Access Modifier service adds the IP address to the *access.conf* with *deny*. It initiates the reconfiguration process rewriting the contents of the *access.conf* file. Example of blocked IP addresses in *access.conf* file are shown in listing 10. Using this strategy, we could stop attacks on the NGINX web server without affecting the server's legitimate users.

3) NGINX dynamic reconfiguration : The default NGINX docker image cannot detect changes in the configuration files on the fly. For this reason, we will create our custom docker image of the NGINX with main *nginx.conf*, *access.conf* for deny listing attacking IP addresses and *nginx-reloader.sh*. The hearth of the custom image is the shell script *nginx-reloader.sh* (see, Listing 11) responsible for dynamic reconfiguration of the NGINX container after the content of the *access.conf* is changed.

HUP is one of the NGINX's control signals responsible for starting new worker processes with a new configuration and graceful shutdown of old worker processes [5]. Triggering this control signal is issued via command *nginx -s reload* after the library *inotify-tools* detects the change in *conf.d* directory.

C. Preventive defense

We leveraged NGINX's built-in features, an important DDoS attack mitigation solution component, as a preventive defense. The main features of this mitigation are as follows [14]:

- **request rate limiting** - we can restrict the number of incoming requests NGINX accepts to a rate typical for normal users. By configuring NGINX, we may limit the client's IP address to access the resource once every two seconds (equivalent to 30 requests per minute). In order to maintain the status of requests for the supplied key (the client IP address), the *limit_req_zone* directive sets up a shared memory zone called *one*,
- **limiting number of connections** - it is also feasible to restrict the number of connections one client's IP address can open. In this case (as in the preceding example), the *limit_conn_zone* directive configures a shared memory zone called *addr* to hold requests for the supplied key,
- **closing slow connections** - not to mention, we can disconnect connections that are writing data too slowly, which may indicate an effort to maintain connections for as long as possible. A prime example of this attack is *Slowloris*. The *client_body_timeout* directive and the *client_header_timeout* directive regulate how long NGINX waits between writing the client body and client headers, respectively.

Listing 12 contains the hardened NGINX configuration we used as a preventive measure in our DDoS defense strategy.

VIII. Evaluation

We discuss the implemented security measure's test results in this section. The locust command was first executed on the client to start imitating actual traffic (see VII-A7). One hundred fifty clients with IP addresses ranging from 172.28.0.100 to 172.28.0.140 made up the valid traffic. One to five seconds passed between each request sent by each client, which each executed on its thread. Each request's destination IP address was 172.28.0.7 (NGINX's IP address). When the z-scores were stable, we launched a DDoS attack using thirty clients connected to the IP 10.10.10.141. The attack was made to repeatedly (every one to three seconds) send requests to the NGINX web server. Then we began tracking metrics in Grafana. First, we observed higher z-score values in our z-score visualizations 6.

Grafana raised an alert to the Supervisor application with a payload containing the IP address (172.28.0.141) after two minutes with no improvement. The attack mitigation process was started right away by the Supervisor application. The current *access.conf* was first fetched, and a check was made to see if the IP was banned. Afterward, it updated the contents in the *access.conf* file and added the IP address 172.28.0.141 to the list of banned IP addresses. From then on, NGINX performed dynamic self-reconfiguration, preventing further access for IP address 172.28.0.141. Figure 8 represents the process

Listing 9 *Grafana alert example.*

```
abs(
  avg(rate(nsnginx_http_response_count_total{app="nginx", status!="403"}[1m])) by (remote_addr)
  - avg (rate(nsnginx_http_response_count_total{app="nginx"}[15m])) by(remote_addr)
) / stddev(rate(nsnginx_http_response_count_total{app="nginx"}[15m])) by (remote_addr)
```

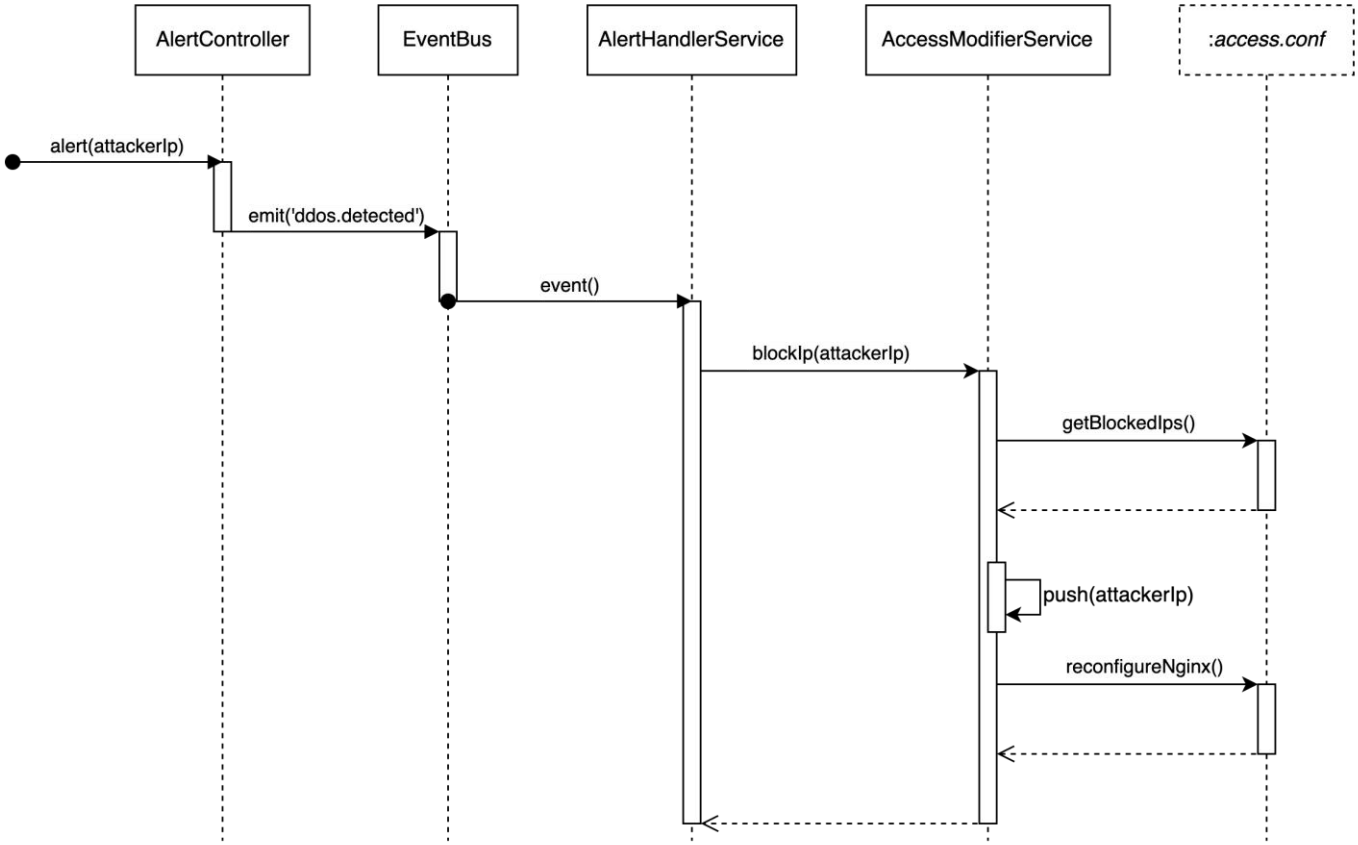

Figure 5: Supervisor mitigation process sequence diagram

Listing 10 *example of blocked IP addresses in access.conf.*

```
deny 172.28.0.109;
deny 172.28.0.111;
deny 172.28.0.101;
```

of NGINX's dynamic reconfiguration on detected *access.conf* change, while the figure 7 depicts logs from the Supervisor application during the attack with all the completed steps.

IX. Conclusion

Identifying application layer DDoS attacks and stopping them without impacting regular users is incredibly difficult. We still need a reliable strategy in place to accomplish this.

In our work, we gave a thorough analysis of the DDoS attacks. There are multiple types of DDoS attacks on multiple RM-OSI layers. In this work, we took it upon ourselves to deeply analyze application layer DDoS attacks that are currently the most challenging. The application layer faces many vulnerabilities like protocol exploits, slow DDoS attacks,

Listing 11 *Shell script for dynamic NGINX reconfiguration.*

```
#!/bin/bash
###########

while true
do
 inotifywait --exclude .swp -e create -e
 ↪  modify -e delete -e move
 ↪  /etc/nginx/conf.d
 nginx -t
 if [ $? -eq 0 ]
 then
  echo "Detected Nginx Configuration Change"
  echo "Executing: nginx -s reload"
  nginx -s reload
 fi
done
```

DNS reflected attacks that indirectly affect web applications, or flood attacks. Each attack has different effects, but they always have unfavorable outcomes, some of which can be fatal to the

Figure 6: Z-score visualization during the DDoS attack

```
[Nest] 18  - 05/10/2023, 1:13:02 AM     LOG [AlertHandlerService] DDoS attack detected
[Nest] 18  - 05/10/2023, 1:13:02 AM     LOG [AccessModifierService] Blocking source IP address of the attack: 172.28.0.141
[Nest] 18  - 05/10/2023, 1:13:02 AM     LOG [AccessModifierService] Reconfiguring nginx access.conf file
[Nest] 18  - 05/10/2023, 1:13:02 AM     LOG [AccessModifierService] IP address has been blocked
```

Figure 7: Supervisor logs during the DDoS attack

Listing 12 *DDoS attack preventive defense NGINX configuration.*

```
http {
    limit_req_zone $binary_remote_addr
↪   zone=one:10m rate=30r/m;
    limit_conn_zone $binary_remote_addr
↪   zone=addr:10m;

    server {
        listen 80;
        listen [::]:80;
        server_name localhost;

        client_body_timeout 5s;
        client_header_timeout 5s;

        location / {
            limit_req zone=one;
            limit_conn_zone
↪   $binary_remote_addr zone=addr:10m;

            root /usr/share/nginx/html;
            index index.html index.htm;
        }
    }
}
```

```
/etc/nginx/conf.d/ MODIFY access.conf
Detected Nginx Configuration Change
Executing: nginx -s reload
```

Figure 8: NGINX logs during the DDoS attack

service provider's reputation.

We have analyzed multiple detection and defense strategies and proposed a concept for attack mitigation using reactive and preventive mechanisms. To the best of our knowledge, we designed a reactive mitigation strategy based on detecting an ongoing attack using web server logs, parsed and exported to a time-series database, on which we performed statistical analysis calculating the z-score. Requests not falling into the normal range were considered malicious, and the system performed the necessary steps. The implemented supervisor application denylisted an attacking IP address, and the custom NGINX container reloaded itself with the new configuration. We utilized NGINX's built-in DDoS defense features for even better security, which may be useful for slow DDoS attacks.

In order to test our solution, we created a dockerized environment with technologies commonly used in real-world scenarios. We presented our successful results and created a working solution that could be implemented in production

environments with few changes.

REFERENCES

[1] PRASEED, Amit; THILAGAM, P. Santhi. DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications. IEEE Communications Surveys & Tutorials, 2018, 21.1: 661-685.

[2] XIE, Yi; YU, Shun-Zheng. Monitoring the application-layer DDoS attacks for popular websites. IEEE/ACM Transactions on networking, 2008, 17.1: 15-25.

[3] ARAFAT, Muhammad Yeasir; ALAM, Muhammad Morshed; ALAM, Mohammad Fakrul. A practical approach and mitigation techniques on application layer DDoS attack in web server. International Journal of Computer Applications, 2015, 131.1: 13-20.

[4] SINGH, Karanpreet; SINGH, Paramvir; KUMAR, Krishan. Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. Computers & security, 2017, 65: 344-372.

[5] "Nginx documentation," NGINX Docs, https://docs.nginx.com/ (accessed May 7, 2023).

[6] "Docker Overview," Docker Documentation, https://docs.docker.com/get-started/overview/ (accessed May 9, 2023).

[7] "Overview: Prometheus," Prometheus Blog, https://prometheus.io/docs/introduction/overview/ (accessed May 9, 2023).

[8] S. Kassabian, "How to use prometheus for anomaly detection in Gitlab," GitLab, https://about.gitlab.com/blog/2019/07/23/anomaly-detection-using-prometheus/ (accessed May 7, 2023).

[9] M. Helmich, "Martin-Helmich/prometheus-nginxlog-exporter: Export metrics from Nginx Access log files to prometheus," GitHub, https://github.com/martin-helmich/prometheus-nginxlog-exporter (accessed May 9, 2023).

[10] "Grafana Documentation: Grafana documentation," Grafana Labs, https://grafana.com/docs/grafana/latest/?pg=oss-graf&plcmt=quick-links (accessed May 9, 2023).

[11] "Documentation: Nestjs - a progressive node.js framework," NestJS, https://docs.nestjs.com/ (accessed May 9, 2023).

[12] "Nginx log metrics [M]," Grafana Labs, https://grafana.com/grafana/dashboards/15947-nginx-log-metrics-m/ (accessed May 9, 2023).

[13] "Locust documentation¶," Locust Documentation - Locust 2.15.1 documentation, https://docs.locust.io/en/stable/ (accessed May 10, 2023).

[14] R. Nelson, "DDos mitigation - using Nginx to prevent ddos attacks," NGINX, https://www.nginx.com/blog/mitigating-ddos-attacks-with-nginx-and-nginx-plus/ (accessed May 10, 2023).

[15] MITTAL, Prateek, et al. Mirage: Towards deployable DDoS defense for Web applications. arXiv preprint arXiv:1110.1060, 2011.

[16] RAHMAN, Md Mahmudur; ROY, Shanto; YOUSUF, Mohammad Abu. DDoS mitigation and intrusion prevention in content delivery networks using distributed virtual honeypots. In: 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT). IEEE, 2019. p. 1-6.

[17] MAHAJAN, Deepika; SACHDEVA, Monika. DDoS attack prevention and mitigation techniques-a review. International Journal of Computer Applications, 2013, 67.19.

[18] DZOGOVIC, Bruno, et al. Zero-Trust Cybersecurity Approach for Dynamic 5G Network Slicing with Network Service Mesh and Segment-Routing over IPv6. In: 2022 International Conference on Development and Application Systems (DAS). IEEE, 2022. p. 105-114.

[19] JAAFAR, Ghafar A., et al. Review of recent detection methods for HTTP DDoS attack. Journal of Computer Networks and Communications, 2019, 2019.

[20] ZEBARI, Rizgar R.; ZEEBAREE, Subhi RM; JACKSI, Karwan. Impact analysis of HTTP and SYN flood DDoS attacks on apache 2 and IIS 10.0 Web servers. In: 2018 International Conference on Advanced Science and Engineering (ICOASE). IEEE, 2018. p. 156-161.

[21] FRÍDEL. Dynamic orchestration of deploying security mechanisms in content delivery networks. In: 2022 Master's thesis, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava.