Implementing a Secure Microservice Architecture: NGINX & Kubernetes Deployment

Orchestration: Alexander Moomaw
Eastern Washington University

Services: Andre Ramirez
Eastern Washington University

Logging: Brendan Hopkins
Eastern Washington University

Frontend: Beighlor Martinez
Eastern Washington University

Defense: Cameron Olivier
Eastern Washington University

Services: Dillon Pikulik
Eastern Washington University

Project Management: Chelsea Edwards
Eastern Washington University

Abstract

This paper presents a secure and scalable deployment architecture for web server environments using Kubernetes. The deployment incorporates a proxy and web application firewall for enhanced security and modular design. Additionally, we introduce our solution to logging and IP banning mechanisms to monitor and restrict unauthorized access. Our approach ensures high availability, load balancing, and robust defense against threats, offering a comprehensive framework for modern web applications.

1 Introduction

Ensuring the security and scalability of web server environments is a challenge in modern digital infrastructure. As cyber threats become more advanced, it's crucial to implement solutions that protect against unauthorized access while also maintaining high availability and performance.

To address these challenges, we propose a comprehensive deployment architecture by first utilizing Kubernetes, an open-source platform designed for automating deployment, scaling, and management of containerized applications. With this platform, we ensure high-availability to prospective users and a modular environment to build and scale upon as we please.

To enhance our microservice architecture, we introduced an additional abstraction layer using an NGINX proxy hosted locally on the server. This proxy server fetches content from our Kubernetes load balancer, serving as a gateway between our internal services and external communications. To mitigate potential threats, we integrated ModSecurity, a well-known web application firewall (WAF), into our proxy. ModSecurity monitors and blocks common exploitation methods by referencing a comprehensive rule list that addresses the OWASP Top Ten vulnerabilities.

We realize that exploiting our endpoint services isn't the only attack vector within our environment. To address this, we implemented host-level logging to monitor network traffic and identify IP addresses not communicating via SSH or HTTP. Additionally, we applied an automated process to block malicious IP addresses, enhancing our overall security posture.

2 Orchestration

The ultimate challenge of any deployment life cycle is tying all ideas into a single, nicely wrapped package. This is the goal of orchestration. To implement, or *orchestrate* a deployment such as this, we designed a multi-tiered architectural diagram (Figure **??**) that outlines how every building block in our environment fits together to form a finalized representation of a working project.

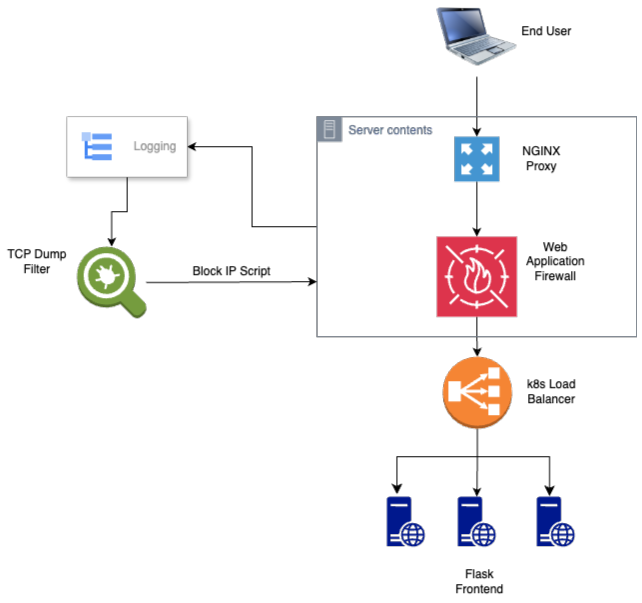

Figure 1: Multi-tiered architectural diagram.

To host our content, we utilized Kubernetes. This method first involved integrating a docker image that was specifically designed to host content created by our frontend developers. We then created service and deployment YAML files to control multiple containers behind a load balancer.

Once we had a functional cluster of orchestrated containers, the next goal was to make our load balancer not the first point of contact between external users and our internal environment. This involved integrating an NGINX reverse proxy on our host machine to facilitate communication without exposing our services directly to the internet.

To setup an NGINX reverse proxy, we had to install the service then modify the global configuration file in the `/etc/nginx/conf.d/` directory to specifically mediate communications between external users and our internal load balancer [**?**]. The specifics for this step is very convoluted, essentially we had to define server and location blocks with a `proxy pass` flag to enable the proxy feature of NGINX.

To tie our scalable, secure environment together on the orchestration end, we installed a web application firewall (WAF) on our reverse proxy to inspect requests being sent to the load balancer [**?**]. The requests will be meticulously cross-referenced against a series of rule lists specifically designed to catch exploits matching the OWASP Top Ten vulnerabilities. If a exploit request is detected, the WAF will drop the request and return a HTTP 403 status code to the threat actor.

3 Logging and IP Gathering

To enhance the security of our web server environment, we implemented a robust mechanism for gathering and logging IP addresses. This approach involves capturing IP addresses attempting to access our server on ports other than 22 (SSH) and 80 (HTTP) and maintaining comprehensive logs for further analysis. The IP gathering mechanism is designed to identify unauthorized access attempts by capturing IP addresses that try to connect to non-standard ports. This is achieved through a script that utilizes tcpdump to monitor network traffic and filter out unwanted connections. The captured IP addresses are stored in a cumulative list, which can be used to block malicious IPs via IP tables. In addition to gathering IP addresses, we implemented an hourly logging mechanism to maintain detailed records of network activity. This involves running a script that logs all IP traffic, excluding ports 22 and 80, and saves the logs with timestamps. These logs provide valuable data for analyzing traffic patterns and identifying potential security threats. To ensure the effectiveness of our IP gathering and logging mechanisms, the captured IP addresses and logs are periodically reviewed. This verification process involves accessing the log files to confirm that the security measures are functioning as intended and to identify any anomalies in network traffic.

4 Defense

We have established a comprehensive security protocol to monitor and block unauthorized access attempts to our web server. This involves capturing IP addresses attempting to access our server on ports other than 22 (SSH) and 80 (HTTP) and maintaining comprehensive logs for further analysis.

The process begins by clearing the log file `/var/log/honeypot/block_ip.log` to ensure new entries are recorded clearly.

The script then checks for the existence of a cumulative IP list file at `/var/log/honeypot/ip_list_all.txt`, which contains IP addresses flagged for suspicious activity. If this list exists, the script reads each IP address and adds a `DROP` rule to `iptables`, effectively blocking incoming traffic from these addresses. The success or failure of each attempt to block an IP address is logged with a timestamp in the `block_ip.log` file.

If the cumulative IP list is not found, an error message is logged and the script exits.

5 Frontend Design

Our Website is called 'Weather and Jokes', and includes four pages: Home, Contact, FTP Manual, and Login. The Home page runs two services, 'Get a Joke' and 'Weather Checking'. The Contact page lists each of our team members and their roles within the group. The FTP Manual page gives instructions to our team members on how to use FTP. The FTP Manual page is meant to be a distraction for threat actors and is not fundamental to the actual Website, but rather a ploy to try and catch attackers attempting to use FTP. The final page, Login, takes in user input and redirects back to the log-in page. The log-in page never actually connects to any backend service, this page is used to try and catch potential threat actors attempting to use attacks such as SQL Injections.

6 Services

References

[1] Hostinger. How to set up nginx reverse proxy. https://www.hostinger.com/tutorials/how-to-set-up-nginx-reverse-proxy/, 2024. Accessed: 2024-05-21.

[2] Linode. Securing nginx with modsecurity. https://www.linode.com/docs/guides/securing-nginx-with-modsecurity/, 2024. Accessed: 2024-05-02.