

Implementing a Secure Microservice Architecture: NGINX & Kubernetes Deployment

Orchestration: Alexander Moomaw
Eastern Washington University

Logging: Brendan Hopkins
Eastern Washington University

Defense: Cameron Olivier
Eastern Washington University

Project Management: Chelsea Edwards
Eastern Washington University

Services: Andree Ramirez
Eastern Washington University

Frontend: Beighlor Martinez
Eastern Washington University

Services: Dillon Pikulik
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 [9]. 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. The 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 [11]. ModSecurity monitors and blocks common exploitation methods by referencing a comprehensive rule list that addresses specific exploits included in the OWASP Top Ten [7]. Exploits that are specifically blocked include SQL injections, cross-site scripting, and command injection [8].

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 1) that outlines how every building block in our environment fits together to form a finalized representation of a working project.

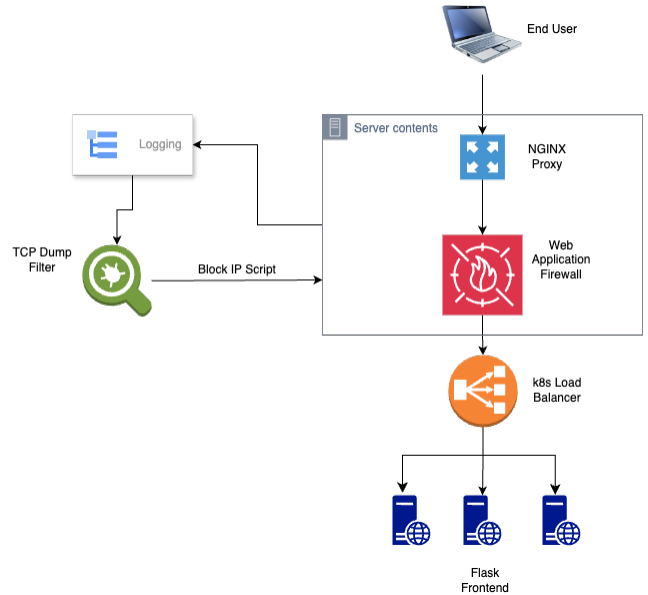


Figure 1: Multi-tiered architectural diagram.

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 [6] 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 [12]. 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 [10]. 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 [5]. 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 [13]. 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

We implemented our website using a Dockerfile pulling an image of Ubuntu which installs Flask that we use as our webserver. You can find more information on backend code and the Flask Server in the services section. We decided to use Flask based on previous experience with this service and Docker so we could containerize our project for added defense. By adding Docker we provided isolation from a singular host and allowed us to add defensive features without effecting our entire system. Our Website is called ‘Weather and Jokes’, and includes four pages: Home, Contact, FTP Manual, and Login. The Home page runs two API 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 and the final page, Login, takes in user input and redirects back to the login page.

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 login page never actually connects to any backend service and is also used to try and catch potential threat actors attempting to use attacks such as SQL Injections. These features allow our webpage to act more as a honeypot gathering information on potential attackers, and allowing our defense team to monitor/log their activities and if necessary block their IP address.

6 Services

We have 2 services, both run in `flask_example.py` file and are presented on the `index.html` page. One of our services is utilizing a weather API. This API has the weather in different geo locations around the world. We use this API to find the weather of a website of a user's choice. The user has a textbox on the `index.html` page that they can enter the website's URL. After they enter the URL they can hit the button to get the weather. This might sound weird at first but it will all make sense. We ask for user input of a website's URL and output the current weather of that website. This is done by taking the URL and running `gethostbyname` to get the IP address of that URL. Now after we get the IP address, we run a `whois` command on the IP address. This command will give us a lot of information on the IP address, but we parse out the physical address associated with the IP. After the physical address is parsed and formatted for the API we call (<https://geocoding.geo.census.gov/>) API and get returned the X, Y geo coordinates of that address [2]. Now that we have geo coordinates we can send those to the weather API. The weather API is <https://api.weather.gov/points/> [3]. After we send the coordinates, we receive the weather of that location, which is the location of the initially given website. All of these commands are automated in the `flask_example.py` file. I chose to implement this service because I have done it in the past and had very well-documented files that aided in quick implementation. This in return aided the rest of the team by speeding up the process of deployment. This service also is very large and can be used for about every website, this could keep the attacks busy, possibly giving information away about themselves.

Another API service that was utilized was the `/joke` route in the Flask application serving a random dad joke to the user. When accessed, the function `joke()` is executed, which sets the API URL <https://icanhazdadjoke.com/> and defines a `cURL` command to request a joke in plain text format [1]. The `subprocess.run` function runs this command and captures the output, which contains the joke. Finally, `result.stdout` is returned, sending the joke back to the user's browser. Additionally, there is a clickable button in `index.html` that allows users to retrieve a joke, providing a simple and interactive way to display a random dad joke. The reason this service was chosen was to have clickable and interactable features on the website so that targets on the website are encouraged to click more and give potential information.

7 Evaluation Methodology

To adequately test our infrastructure, we set out to exploit the specific methods that the web application firewall was built to defend against. This is done through the login page of our frontend environment. A successful evaluation will be

represented by any attack execution being redirected via a HTTP 403 response, indicating that the attack was successfully detected and blocked by our web application firewall.

The first exploit we chose was an SQL injection, specifically a tautology attack. This attack aims to break the conditional statement for a typical user name/password login combination on an SQL backend server such that when the query is made through the login page to check if the user exists the condition will always be true, thereby allowing the attacker to log in as any user that exists [14].

The second exploit we chose was a cross-site scripting (XSS) attack. This attack attempts to inject malicious scripts into input sections or source code of a website [15]. This attack will also be tested by appending an exploit to the end of our URL. If the threat actor is able to successfully execute their attack, they would be able to run arbitrary scripts in the context of the user's browser. This could lead to a range of malicious activities, such as stealing cookies, session tokens, or other sensitive information, defacing the website, or redirecting users to malicious sites. Our testing will involve various payloads to simulate real-world XSS attack vectors and assess the vulnerability of our web application to these types of exploits.

The final exploit we will be testing is command injection. This exploit attempts to bypass an input field in such a way that allows the attacker to execute arbitrary operating system commands on the server [16]. If an attacker is able to successfully perform this attack, they can compromise the web server, exfiltrate data, and establish persistence by injecting a backdoor into our system [4].

8 Results

Results

References

- [1] Dad jokes api. Available at: <https://icanhazdadjoke.com/>.
- [2] Geo location api. Available at: <https://geocoding.geo.census.gov/>.
- [3] Weather api. Available at: <https://api.weather.gov/points/>.
- [4] DRD. Use command injection to pop a reverse shell on a web server, 2018. <https://null-byte.wonderhowto.com/how-to/use-command-injection-pop-reverse-shell-web-server-01>
- [5] Damon Garn. How to capture and analyze traffic with tcpdump, 2023. <https://www.techtarget.com/searchnetworking/tutorial/How-to-capture-and-analyze-traffic-with-tcpdump>.

- [6] Hostinger. nginx is an http and reverse proxy server. <https://www.hostinger.com/tutorials/how-to-set-up-nginx-reverse-proxy/>, 2024. Accessed: 2024-05-21.
- [7] <https://owasp.org/>. The owasp top 10 is a standard awareness document for developers and web application security. <https://owasp.org/www-project-top-ten/>, 2021. Accessed: 2024-05-21.
- [8] <https://owasp.org/>. Owasp modsecurity core rule set. <https://owasp.org/www-project-modsecurity-core-rule-set/>, 2024. Accessed: 2024-05-02.
- [9] Kubernetes. This page is an overview of kubernetes. <https://kubernetes.io/docs/concepts/overview/>, 2023. Accessed: 2024-05-21.
- [10] Linode. Securing nginx with modsecurity. <https://www.linode.com/docs/guides/securing-nginx-with-modsecurity/>, 2024. Accessed: 2024-05-02.
- [11] modsecurity. This page is an overview of modsecurity. <https://github.com/owasp-modsecurity/ModSecurity>, 2021. Accessed: 2024-05-21.
- [12] Nginx. How to set up nginx reverse proxy. <https://nginx.org/en/>, 2024. Accessed: 2024-05-21.
- [13] Daniel Schwartz. Schedule tcpdump with cron, 2018. Available at: <https://schwartzdaniel.com/schedule-tcpdump-with-cron/>.
- [14] Chandershekhar Sharma. Types of sql manipulation, 2016. https://www.researchgate.net/figure/Types-of-SQL-manipulation-a-Tautology-In-this-type-of-SQLIA-an-attacker-exploits-an_fig4_299575353.
- [15] Synopsys. Cross-site scripting (xss), 2022. [https://www.synopsys.com/glossary/what-is-cross-site-scripting.html#:~:text=Cross%2Dsite%20scripting%20\(XSS\)%20is%20an%20attack%20in%20which,the%20user%20to%20click%20it](https://www.synopsys.com/glossary/what-is-cross-site-scripting.html#:~:text=Cross%2Dsite%20scripting%20(XSS)%20is%20an%20attack%20in%20which,the%20user%20to%20click%20it).
- [16] Weilin Zhong. Command injection, 2022. https://owasp.org/www-community/attacks/Command_Injection#:~:text=Command%20injection%20is%20an%20attack.