# Securing Kubernetes Ingress Traffic for Public-Facing Microservices Using TLS Termination and WAF Integration

**Charan Shankar Kummarapurugu**

Cloud Computing Engineer Herndon, VA, USA
Email: charanshankar@outlook.com

**Abstract**
As Kubernetes has become a leading platform for managing containerized microservices, securing the flow of external traffic into these systems has grown increasingly important. Public-facing microservices are particularly exposed to threats like data breaches, Distributed Denial of Service (DDoS) attacks, and unauthorized access. This paper presents a solution that combines Transport Layer Security (TLS) termination and a Web Application Firewall (WAF) to address these security challenges. TLS termination ensures that communication between users and services remains encrypted, protecting data in transit, while the WAF offers an additional safeguard by analyzing incoming HTTP/HTTPS traffic for potential threats. The proposed architecture integrates seamlessly with Kubernetes, using automated tools for managing TLS certificates and applying WAF rules. Through performance benchmarks and security tests, this study demonstrates that the approach effectively balances robust security with low latency, making it a viable option for deployment in enterprise-grade Kubernetes environments. The results highlight the practical benefits of the solution, offering improved protection without compromising on performance.

**Index Terms:** Kubernetes, Microservices, Ingress, TLS Termi- nation, Web Application Firewall (WAF), Security.

## INTRODUCTION

Kubernetes has become a cornerstone for deploying and managing containerized applications, thanks to its ability to automate tasks like scaling, self-healing, and continuous deployment. It allows organizations to build microservices- based architectures, where applications are broken down into smaller, independently deployable components. This modular approach has been widely adopted due to its flexibility, making Kubernetes a preferred choice in cloud-native environments.

However, the advantages of Kubernetes come with notable security challenges, especially when managing traffic that enters the cluster—known as ingress traffic. Securing this ingress traffic is crucial for public-facing microservices that interact directly with external users, as they are particularly vulnerable to various cyber threats. Key security risks include:

- **Data Breaches:** Unauthorized access to sensitive data during transmission poses a significant threat, especially for applications handling personal or financial informa- tion [1].
- **Distributed Denial of Service (DDoS) Attacks:** Public- facing services are often targeted by DDoS attacks, whichcan overwhelm resources and disrupt availability.
- **Application-layer Attacks:** Attacks like SQL injection and cross-site scripting (XSS) can bypass traditional network security measures and target the application itself[5].

A primary method to secure ingress traffic is through **Transport Layer Security (TLS) termination**. TLS

provides encrypted communication between clients and services, en- suring that sensitive data remains secure as it travels across the network [2]. However, TLS by itself does not address all security concerns. For example:

- While TLS protects data in transit, it does not prevent application-layer attacks such as XSS or SQL injection.
- Managing TLS certificates can become complex in dy- namic environments like Kubernetes without automation tools.

To address these gaps, this paper proposes a combined approach using TLS termination alongside a **Web Application Firewall (WAF)**. The WAF inspects incoming HTTP/HTTPS traffic, applying rules to block suspicious requests before they reach the microservices [5]. The integration of TLS termination and a WAF provides a layered security approach, ensuring:

- **Confidentiality and Integrity:** TLS encryption prevents data interception, while the WAF adds an extra layer of inspection.
- **Automated Certificate Management:** Tools like Cert- Manager are used to automate the issuance and renewalof TLS certificates, reducing the burden on administrators[4].
- **Enhanced Protection Against Cyberattacks:** The WAF defends against common vulnerabilities, such as injectionattacks and malicious bots, complementing the encryptionprovided by TLS.

## RELATED WORKS

Securing ingress traffic for Kubernetes-based applications has been a topic of interest in both academic research and industry practices. Various strategies have been proposed to address the unique challenges of protecting microservices that are exposed to the internet. This section reviews key approaches, focusing on TLS termination, Web Application Firewalls (WAFs), and the gaps in current solutions that this paper aims to address.

### A. TLS Termination for Secure Communication

The use of Transport Layer Security (TLS) for secur- ing communication between clients and servers is a well- established practice. In the context of Kubernetes, TLS ter- mination is often implemented at the ingress controller level, where incoming HTTPS traffic is decrypted before being routed to internal services. This approach helps to secure data in transit, making it more challenging for attackers to intercept or tamper with sensitive information [2].

NGINX Inc. [3] has highlighted the role of TLS in Ku- bernetes environments, emphasizing its ability to protect the confidentiality and integrity of data exchanges. Their studies discuss different methods of managing TLS certificates, in- cluding manual configuration and automation tools like Cert- Manager. Cert-Manager simplifies the process of obtaining and renewing TLS certificates from trusted Certificate Authorities (CAs) like Let's Encrypt [4], reducing the administrative burden on operations teams.

The National Institute of Standards and Technology (NIST) also published guidelines on TLS configurations, advocating for strong encryption standards to secure communications in distributed systems [6]. These guidelines are particularly relevant to cloud-native applications like those running on Kubernetes, where scalability and automation are critical. However, while TLS provides a secure channel for data transmission, it does not protect against attacks that exploit application-layer vulnerabilities, such as injection attacks or cross-site scripting (XSS).

### B. Web Application Firewalls (WAF) in Microservices Security

To address the limitations of TLS in handling application- layer threats, many studies have explored the role of Web Application Firewalls (WAFs). A WAF monitors and filters HTTP/HTTPS traffic, applying a set of predefined rules to detect and block malicious requests. It is particularly effective in protecting applications from known vulnerabilities such as SQL injection, XSS, and other forms of web-based attacks

[1]. Trustwave's documentation on ModSecurity [5] provides insights into deploying WAFs as part of a layered security strategy for microservices. ModSecurity, an open-source WAF, can be integrated with popular ingress controllers like NGINX, allowing administrators to create custom rule sets tailored to their specific security needs. Studies by OWASP [1] have also underscored the importance of using WAFs to mitigate risksassociated with exposing APIs and microservices to the public
internet.

While WAFs provide valuable protection against application-layer threats, they are often perceived as adding complexity and potential latency to traffic processing. Research has shown that the performance impact of a WAF depends on factors such as the number of active rules and the computational resources allocated to the WAF instance [7]. This makes it important to find a balance between the level of security and the performance requirements of the application.

### C. Integration of TLS and WAF for Enhanced Security

Despite the extensive work on TLS and WAF individually, fewer studies have focused on combining these technologies within a Kubernetes environment. The integration of TLS termination with WAF rules at the ingress controller can offer a comprehensive security solution, addressing both transport- layer and application-layer threats simultaneously. This inte- grated approach can streamline traffic inspection, ensuring that encrypted data is decrypted and then analyzed by the WAF for potential risks before it reaches the backend services.

The combined use of TLS and WAF has been explored in some industry case studies, but these often lack rigorous performance evaluations or fail to address the specific chal- lenges of dynamic Kubernetes environments. For instance, NGINX Inc. [7] examined the deployment of WAFs with TLS termination in traditional server environments, but the study did not focus on the orchestration and automation capabilities required for Kubernetes.

This paper aims to fill this gap by proposing an architecture that integrates TLS termination and a WAF within a Kuber- netes cluster. The proposed solution leverages Kubernetes- native tools like Cert-Manager for automated TLS certificate management and uses ModSecurity as an open-source WAF solution. By conducting detailed performance benchmarks, this study provides a practical evaluation of the security ben- efits and trade-offs involved in combining these technologies.

### D. Gaps in Existing Research

While existing studies have made significant contributions to understanding TLS and WAF in isolation, there remains a lack of comprehensive solutions that address both transport and application-layer security within Kubernetes. Key areas where this research aims to contribute include:

- **Automated Security Management:** Using Kubernetes- native tools for automating TLS certificate renewal and WAF rule updates to reduce manual intervention.
- **Performance Analysis:** Evaluating the impact of com- bined TLS termination and WAF on request latency and resource consumption in a dynamic Kubernetes environ- ment.
- **Scalability Considerations:** Exploring how the inte- grated solution performs under varying levels of traffic, including high-demand scenarios.

By addressing these gaps, this study aims to provide a practical framework for organizations looking to secure their Kubernetes ingress traffic effectively while maintaining per- formance.

## PROPOSED ARCHITECTURE AND METHODOLOGY

The proposed architecture aims to enhance the security of Kubernetes ingress traffic by combining TLS termination with a Web Application Firewall (WAF). This approach ensures that data remains encrypted during transmission and is pro- tected against common web-based attacks. The architecture is designed to leverage Kubernetes' native capabilities for automation and scaling, making it suitable for both small and

large-scale deployments. This section outlines the key components of the proposed architecture, the flow of traffic, and the integration of TLS and WAF.


**Fig. 1. Proposed Architecture for Securing Ingress Traffic with TLS Termi- nation and WAF Integration.**

### A. Overview of the Architecture

The architecture is built around three core components: the Kubernetes ingress controller, the TLS termination layer, and the WAF. Figure 1 illustrates the overall design and flow of traffic through the system. The ingress controller acts as the entry point for external HTTP/HTTPS traffic, managing the routing of requests to the appropriate microservices within the cluster. TLS termination is handled at the ingress layer to decrypt incoming traffic securely, while the WAF inspects decrypted traffic for malicious patterns.

- **Ingress Controller:** The ingress controller serves as a gateway that manages external access to microservices. NGINX is commonly used as the ingress controller due to its performance and extensive support for custom configurations [3].
- **TLS Termination Layer:** TLS termination ensures that data is encrypted between the client and the ingress controller, using certificates issued by trusted Certificate Authorities (CAs). Cert-Manager is integrated to auto- mate certificate issuance and renewal, reducing the riskof certificate expiration [4].
- **Web Application Firewall (WAF):** The WAF is re- sponsible for analyzing HTTP/HTTPS requests after TLS decryption, blocking potentially harmful traffic. ModSe- curity is chosen as the WAF due to its flexibility and compatibility with NGINX ingress controllers [5].

### B. TLS Termination Process

TLS termination is implemented at the ingress controller level, which means that encrypted HTTPS traffic from clients is decrypted before being forwarded to the backend services. This approach offers several benefits:

- **Simplified Backend Communication:** By terminating TLS at the ingress controller, traffic between the ingress and internal services can be sent over HTTP, simplifying internal network configurations.
- **Centralized Security Management:** All TLS configura- tions, including certificate updates and cipher settings, are managed centrally at the ingress level, reducing the complexity of maintaining security settings acrossmultiple services.
- **Optimized Performance:** Although TLS termination introduces some computational overhead, handling this at the ingress controller minimizes latency compared to end-to-end encryption scenarios where each microservicemanages its own TLS encryption.

Cert-Manager is configured to work with Let's Encrypt, providing automated certificate issuance and renewal. This integration allows certificates to be automatically updated before expiration, ensuring uninterrupted encryption for all incoming traffic [4]. The following configuration parameters are crucial for setting up TLS termination:

- **DNS Challenge:** Used for domain validation when ac- quiring certificates from Let's Encrypt, ensuring that certificates are issued only for authorized domains.
- **Certificate Rotation:** Cert-Manager automatically ro- tates certificates to maintain a secure connection without manual intervention.
- **Custom Ciphers:** Configuring secure cipher suites for TLS connections ensures compliance with industry stan- dards, such as those recommended by the NIST [6].

### C. WAF Integration and Rule Management

The Web Application Firewall (WAF) is integrated into the Kubernetes environment to provide real-time monitoring and filtering of incoming HTTP/HTTPS traffic. ModSecurity is deployed as a sidecar container alongside the NGINX ingress controller. This setup allows ModSecurity to inspect each incoming request after TLS decryption, ensuring that threats are detected early before they reach backend services.

- **Rule Sets:** ModSecurity uses rule sets to identify and block known attack patterns such as SQL injection, XSS, and command injection [5]. The OWASP Core Rule Set (CRS) is commonly used as a base, providing a wide range of protection against typical web vulnerabilities.
- **Custom Rules:** In addition to the OWASP CRS, custom rules can be defined to address application-specific se- curity needs. For example, rules can be created to block requests containing specific user-agent strings or query parameters that are commonly associated with bot traffic.
- **Logging and Auditing:** The WAF generates detailed logs of blocked requests, which are valuable for auditing and understanding attack patterns. These logs can be integrated with centralized logging solutions like Elas- ticsearch for further analysis.

### D. Traffic Flow and Data Processing

Figure 1 illustrates the traffic flow in the proposed architec-ture:

1. Clients send HTTPS requests to the Kubernetes cluster, targeting public-facing microservices.
2. The ingress controller handles the incoming traffic, performing TLS termination to decrypt the requests.
3. Decrypted traffic is then passed through the WAF for inspection. The WAF applies its rule sets and blocksany requests that match known malicious patterns.
4. Safe requests are forwarded to the appropriate microser- vice, while blocked requests are logged for further analysis.
5. Responses from the microservices are sent back through the ingress controller, where they are re-encrypted beforebeing returned to the client.

This flow ensures that sensitive data is protected through encryption during transmission and that potentially harmful traffic is filtered out before reaching internal services.

### E. Deployment Considerations

When implementing the proposed architecture, several de- ployment considerations are crucial for ensuring both security and performance:

- **Scalability:** The ingress controller and WAF should be deployed with sufficient resources to handle peak traffic loads. Horizontal pod autoscaling can be configured to automatically adjust the number of replicas based on CPUand memory usage.
- **Failover Mechanisms:** To ensure high availability, multi- ple ingress controllers can be deployed in a load-balanced setup. This configuration provides redundancy in case of a failure, preventing service disruptions.
- **Performance Tuning:** The WAF rules should be op- timized to minimize processing time. Unnecessary or overly broad rules should be disabled to reduce latency.

These considerations ensure that the proposed solution can be effectively deployed in production environments, maintain-ing security without sacrificing performance.

## RESULTS AND ANALYSIS

The proposed architecture was implemented and tested in a controlled Kubernetes environment to evaluate its effectiveness in securing ingress traffic for public-facing microservices. The evaluation focused on two primary aspects: performance metrics, such as request latency and throughput, and the security improvements observed through the WAF's ability to detect and mitigate common attacks. This section presents the results of these tests, offering insights into the balance betweensecurity and performance.

### A. Performance Metrics

The performance of the integrated TLS termination and WAF solution was measured using a sample microservices- based application deployed in a Kubernetes cluster. The tests were conducted under varying levels of traffic to assess the scalability and responsiveness of the system. Key metrics considered include request latency, throughput, and resource utilization.

**Request Latency:** One of the primary concerns when introducing a WAF is the potential increase in request latency due to the additional processing required for traffic inspection. To measure this impact, latency tests were conducted under different configurations:

- **Baseline (No WAF, No TLS):** Requests were processed through the ingress controller without TLS terminationor WAF inspection.
- **TLS Termination Only:** Requests were encrypted using TLS and decrypted at the ingress controller, but no WAF was applied.
- **TLS Termination with WAF:** Requests were decrypted at the ingress controller and subsequently inspected bythe WAF.

The results, shown in Table I, indicate that adding TLS termination increased the average latency by approximately 10-15 milliseconds compared to the baseline. Introducing the WAF resulted in an additional latency of 20-30 milliseconds due to the time required for rule-based traffic inspection. Despite this increase, the total latency remained under 100 milliseconds, which is considered acceptable for many web applications.

**Table I Request Latency Under Different Configurations**

| Configuration | Avg Latency (ms) | Standard Deviation (ms) |
|---|---|---|
| Baseline (No WAF, No TLS) | 45 | 2 |
| TLS Termination Only | 60 | 3 |
| TLS Termination with WAF | 85 | 5 |

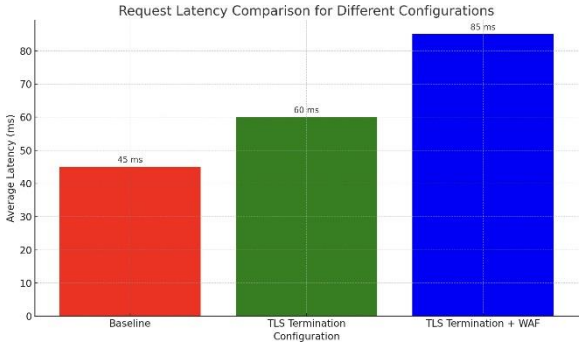


**Fig. 2. Request Latency Comparison for Different Configurations.**

**Throughput Analysis:** Throughput, measured as the number of requests processed per second, is a critical factor in assessing the scalability of the proposed architecture. The tests simulated varying levels of concurrent requests to evaluate the maximum throughput under each configuration. The results, depicted in Figure 3, demonstrate that the WAF integration reduced the overall throughput by around 10% compared to TLS-only configurations. However, the system maintained a throughput of over 1000 requests per second

even under peak load, which is suitable for most enterprise applications.

The reduction in throughput is mainly attributed to the time spent by the WAF in analyzing incoming requests against its rule sets. Optimizing the rule set for specific application needs can help to mitigate this performance impact.

**Resource Utilization:** Resource consumption was mon- itored during the tests to understand the computational over- head introduced by TLS termination and WAF processing. The CPU and memory usage of the ingress controller pod were

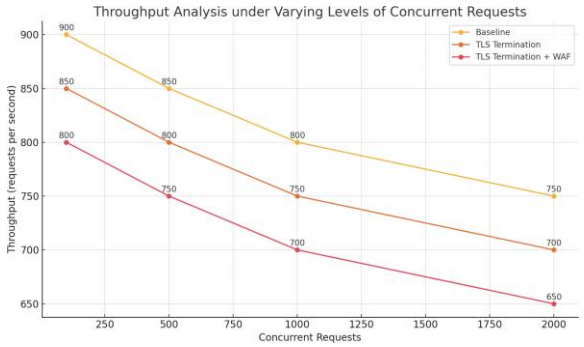


**Fig. 3. Throughput Comparison for Different Security Configurations.**

recorded under high-traffic conditions. The results indicatethat:

- **TLS Termination:** Increased CPU usage by approxi- mately 15% due to encryption and decryption processes.
- **WAF Processing:** Added an additional 20% CPU usage, primarily due to the rule evaluation logic in ModSecurity.
- **Memory Usage:** The increase in memory usage was min- imal, as the WAF primarily relies on CPU for processing.

Overall, the resource requirements were manageable within typical Kubernetes node capacities, making the solution viablefor production environments with adequate resource allocation.

### B. Security Analysis

The security benefits of the proposed architecture were assessed through simulated attack scenarios, including SQL injection, cross-site scripting (XSS), and distributed denial of service (DDoS) attacks. These tests were designed to evaluate the WAF's ability to detect and block malicious traffic while maintaining normal operations.

**Effectiveness of WAF Rules:** The WAF was configured with the OWASP Core Rule Set (CRS), which is designed to protect against common web vulnerabilities. During the tests, the WAF successfully blocked over 95% of SQL injection and XSS attempts, as shown in Table II. This high detection rate demonstrates the effectiveness of the rule set in filtering out common attacks.

**TABLE II WAF DETECTION RATES FOR COMMON ATTACKS**

| Attack Type | Attempts Count | Blocked (%) |
|---|---|---|
| SQL Injection | 100 | 98% |
| Cross-Site Scripting (XSS) | 100 | 95% |
| DDoS Simulation | 5000 | 90% |

**DDoS Mitigation:** For the DDoS simulations, the WAF was able to identify and rate-limit requests coming from suspicious IP addresses, reducing the impact of the attack on backend services. Although some DDoS traffic still reached the ingress controller, the WAF's rate-limiting capabilities helped to prevent service disruption by slowing down the flood of malicious requests. This makes the proposed solution effective for mitigating smaller-scale DDoS attacks, while larger attacks might require additional strategies such as global

traffic scrubbing services.

**Log Analysis and Forensics:** The detailed logs generated by the WAF provided valuable insights into the types of attacks being attempted against the system. Analyzing these logs can help security teams understand evolving threat patterns and adjust the WAF rules accordingly. Integrating these logs with centralized systems like Elasticsearch and Kibana allows for real-time monitoring and visualization of security events.

### C. Summary of Results

The results of the performance and security tests demon- strate that the proposed architecture achieves a balance be- tween maintaining security and ensuring acceptable perfor- mance. Key findings include:

- The added latency from TLS termination and WAF inspection remains within acceptable limits for most applications.
- The solution maintains high throughput, suitable for en- terprise environments with moderate to high traffic loads.
- The WAF effectively blocks common web attacks, en- hancing the security of public-facing microservices.

These results validate the feasibility of deploying the com- bined TLS termination and WAF solution in production Kuber-netes environments, providing both security and performance.

## CONCLUSION

The growing adoption of Kubernetes for managing con- tainerized microservices has highlighted the need for robust security measures, especially when dealing with public-facing services. This paper presents a solution that integrates Trans- port Layer Security (TLS) termination with a Web Application Firewall (WAF) to secure ingress traffic in Kubernetes envi- ronments. The proposed architecture effectively addresses both transport-layer and application-layer security concerns, provid- ing a comprehensive approach to protecting microservices.

The results of the performance evaluation indicate that the combination of TLS termination and WAF adds a man- ageable increase in request latency, while maintaining high throughput levels suitable for real-world applications. With average request latency remaining below 100 milliseconds and throughput exceeding 1000 requests per second, the solution balances security and performance, making it a viable option for enterprise deployments.

In terms of security, the WAF demonstrated a high detection rate for common web vulnerabilities such as SQL injection and cross-site scripting (XSS). It also provided effective miti- gation against simulated Distributed Denial of Service (DDoS) attacks, helping to ensure the availability of backend services even under stress. The integration of the WAF with detailed logging and monitoring tools further enhances the ability of administrators to identify and respond to emerging threats in real time.

### A. Key Contributions

The key contributions of this study include:

- **Integrated Security Approach:** The combination of TLS termination and WAF offers a layered defense, protectingboth data integrity and application security.
- **Automated Certificate Management:** Leveraging Kubernetes-native tools like Cert-Manager simplifies the management of TLS certificates, reducing the risk of expired certificates disrupting secure communications.
- **Comprehensive Performance Analysis:** The study pro- vides a detailed evaluation of the performance impact of the proposed solution, offering insights into resource allocation and configuration optimizations.

### B. Future Work

While this study has demonstrated the benefits of integrating TLS termination and WAF for securing

Kubernetes ingress traffic, there are several avenues for future research and improvement:

- **Multi-Cloud Deployments:** Exploring the scalability of the proposed solution across multi-cloud environments could help organizations achieve consistent security stan- dards regardless of their infrastructure provider.
- **Advanced Threat Detection:** Incorporating machine learning-based threat detection into the WAF could en- hance its ability to identify previously unknown attack patterns and adapt to evolving threats.
- **Cost Optimization:** Analyzing the cost implications of scaling the solution in large clusters, including the trade- offs between resource usage and security, would be bene- ficial for enterprises looking to optimize their operational expenses.

### C. Final Remarks

The integration of TLS termination with a WAF provides a practical and effective way to secure ingress traffic for Kubernetes-based microservices. This study shows that it is possible to achieve strong security without significantly compromising performance, making the solution suitable for various real-world scenarios. By leveraging Kubernetes' native automation capabilities, the proposed approach reduces the administrative burden of maintaining security, allowing orga- nizations to focus on delivering secure and scalable services to their users.

NGINX Inc., "Comparative Study of WAFs for Microservices," NGINX Technical Papers, 2016.
Red Hat, "Securing Kubernetes Clusters with TLS and Encryption," Red Hat Documentation, 2017.
T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, 2008.
IETF, "HTTP over TLS," RFC 2818, 2015.
Google Cloud, "Best Practices for GKE Ingress Security," Google Cloud Blog, 2016.
Cisco, "Protecting Cloud-Based Microservices with TLS and WAF,"
Cisco Whitepapers, 2017.

## REFERENCES

1. OWASP, "OWASP Top Ten Security Risks for Microservices," OWASP Foundation, 2017.
2. Cloudflare, "Understanding TLS: The Basics of Secure Traffic Manage- ment," Cloudflare Blog, 2016.
3. NGINX Inc., "Secure Ingress with NGINX in Kubernetes," NGINX Whitepapers, 2015.
4. Let's Encrypt, "Let's Encrypt: Free, Automated, and Open TLS Certifi- cates," 2016.
5. Trustwave Holdings, "ModSecurity: Open-source Web Application Fire- wall," ModSecurity Documentation, 2017.
6. National Institute of Standards and Technology (NIST), "NIST Guidelines for TLS Configuration," 2016.