Chapter 1
Introduction

In the contemporary digital era, the omnipresence of web applications has transformed how businesses operate, individuals communicate, and services are delivered. However, with the exponential growth in the deployment of these web applications comes an equally significant rise in security vulnerabilities and cyber threats. Ensuring the security of web applications is paramount to safeguarding sensitive data, maintaining user trust, and ensuring the uninterrupted operation of online services. The aim is to address these security concerns by implementing a robust Web Application Firewall (WAF) leveraging Nginx and ModSecurity, integrated with threat intelligence feeds to enhance its defensive capabilities.

Web Applications

Web applications have become crucial tools in various sectors, including finance, healthcare, education, and e-commerce. They facilitate seamless interactions and transactions, offering convenience and efficiency. However, this convenience comes at a cost. Web applications are often targeted by malicious actors seeking to exploit vulnerabilities for personal gain, espionage, or disruption of services. Common web application attacks include SQL injection, cross-site scripting (XSS), and distributed denial of service (DDoS) attacks. These attacks can lead to data breaches, financial losses, and reputational damage.

The traditional approach to web security often involves periodic vulnerability assessments and penetration testing. While these measures are crucial, they are reactive in nature. There is a growing need for proactive defenses that can detect and mitigate threats in real-time. This is where Web Application Firewalls (WAFs) come into play. WAFs are designed to protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. They are capable of blocking malicious traffic and preventing attacks before they reach the application.

Web Application Firewalls (WAFs)

A Web Application Firewall (WAF) is a security system designed to detect and prevent attacks targeting web applications. Unlike traditional firewalls that guard the perimeter of a network, WAFs are specifically tailored to protect the application layer, which is often where the most sensitive and valuable data resides. WAFs can detect and block varioustypes of attacks, such as:

1. **SQL Injection**: Malicious SQL queries are injected into input fields to manipulate the database.
2. **Cross-Site Scripting (XSS)**: Malicious scripts are injected into web pages viewed by other users.
3. **Cross-Site Request Forgery (CSRF)**: Unauthorized commands are transmitted from a user that the web application trusts.
4. **File Inclusion**: Attackers include a file from an external source in the web server.

By inspecting incoming traffic, a WAF can identify and block suspicious activities, ensuring that only legitimate requests are processed by the web application.

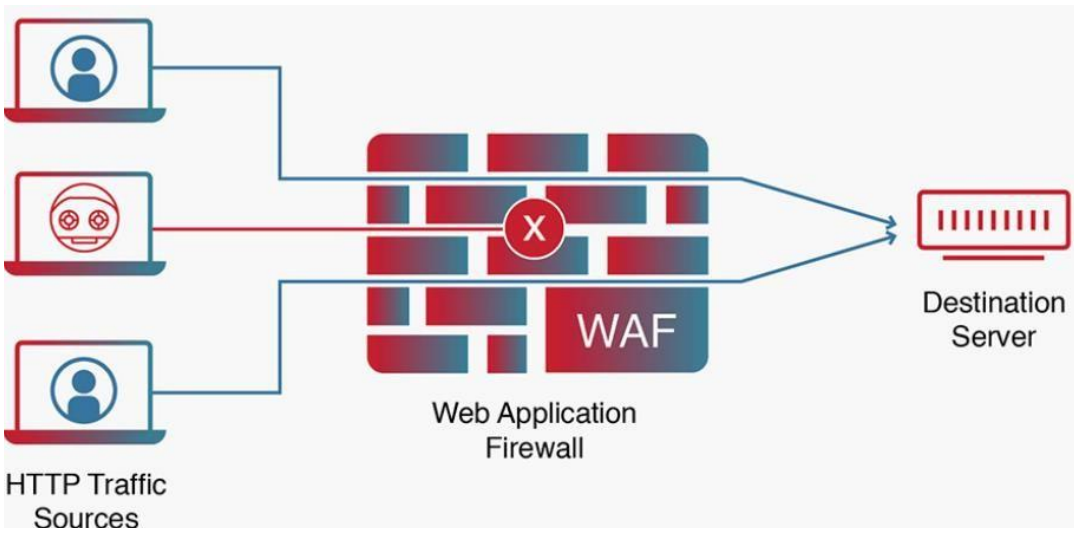


Figure 1.1: Web Application Firewall (WAF) Protection from Threats

The figure 1.1 illustrates the flow of HTTP traffic from various sources to a destination server, with an added layer of security provided by a Web Application Firewall (WAF). The HTTP traffic sources, which can include users' web browsers, mobile apps, or other systems, send requests to the destination server. However, before reaching the server, the traffic passes through the WAF, a security system designed to filter, monitor, and block malicious traffic. The WAF inspects and analyses the incoming traffic, blocking any suspicious or malicious requests, and only allowing legitimate traffic to pass through. Once cleared by the WAF, the traffic reaches the destination server, which processes the requests and sends responses back to the traffic sources. By sitting between the traffic sources and the destination server, the WAF acts as a protective barrier, shielding the server from potential security threats and ensuring that only authorized traffic reaches the server.

1.1 Nginx and ModSecurity WAF

Nginx is a highly efficient and versatile web server and reverse proxy server. It is known for its high performance, scalability, and ability to handle a large number of concurrent connections. Nginx is widely used in the industry, powering many high-traffic websites and applications. Its modular architecture allows for the integration of additional functionalities through modules, making it an ideal candidate for building a WAF.

ModSecurity (ModSec) is an open-source web application firewall (WAF) designed to protect web applications by inspecting HTTP traffic and applying customizable rules to detect and mitigate attacks such as SQL injection and cross-site scripting (XSS). Initially an Apache module, it now supports nginx and IIS, offering real-time monitoring, logging, and filtering for enhanced web security.

Integrating ModSecurity with nginx combines ModSecurity's robust security features with the high performance and efficiency of nginx. This integration allows web administrators to leverage ModSecurity's powerful rule-based protection mechanisms within the nginx environment, providing a secure, scalable, and efficient solution for protecting web applications against a wide range of cyber threats.

1.2 Threat Intelligence Integration

Threat intelligence refers to the collection and analysis of information about current and emerging cyber threats. Integrating threat intelligence with a WAF enhances its ability to detect and mitigate new and sophisticated attacks. Threat intelligence feeds provide real-time data on known malicious IP addresses, URLs, and attack patterns. By incorporating this data into the WAF, it becomes capable of identifying and blocking threats that may not yet be covered by its static rule set.

The integration of threat intelligence with ModSecurity involves continuously updating the WAF's rule set based on the latest threat data. This dynamic approach ensures that the WAF remains effective against evolving threats. Automated scripts can be employed to extract rules from threat intelligence feeds, convert them into a format compatible with ModSecurity, and update the WAF configuration accordingly.

The digital landscape today is marked by an unprecedented reliance on web applications for a multitude of services spanning across various sectors such as finance, healthcare, education, and e-commerce. These applications serve as the backbone of modern business operations, facilitating seamless interactions and transactions that drive efficiency and growth. However, the proliferation of web applications has been accompanied by a parallel increase in cyber threats, making web security a critical concern for organizations worldwide.

Web applications are particularly vulnerable to a range of attacks, including SQL injection, cross-site scripting (XSS), and distributed denial of service (DDoS) attacks. These attacks exploit vulnerabilities in web applications to gain unauthorized access to data, disrupt services, and inflict financial and reputational damage. The need for robust security measures to protect web applications has never been more pressing.

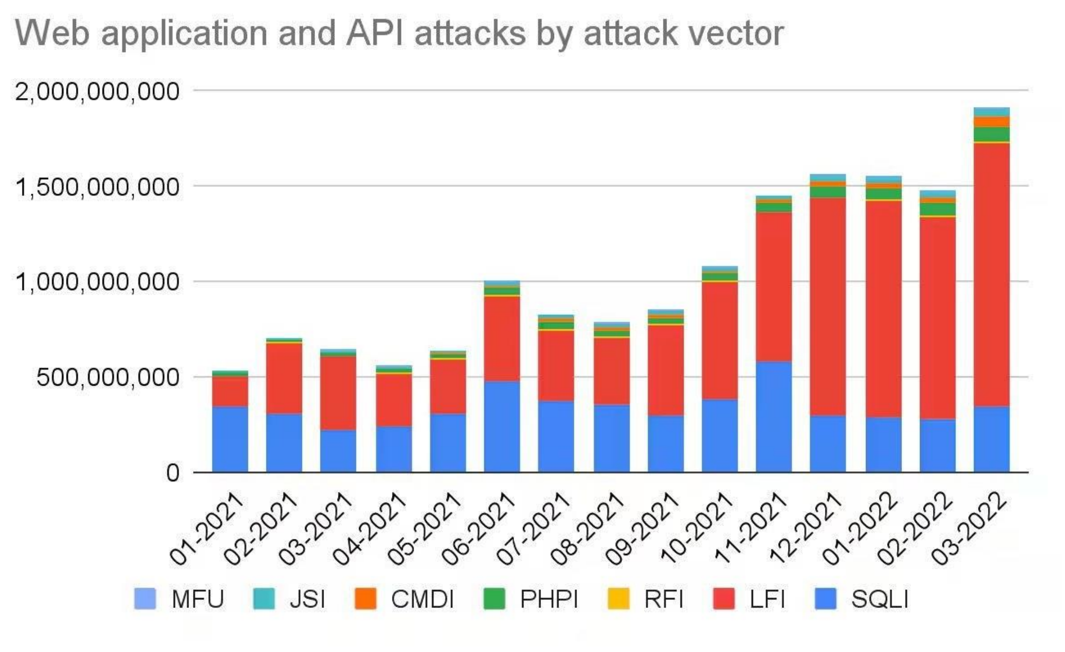


Figure 1.2: Increase Rate in Web-based Attacks during 2021 and 2022

The bar chart titled "Web application and API attacks by attack vector" in the figure 1.2 illustrates a significant increase in the number of attacks from January 2021 to March 2022, with notable spikes in June 2021, December 2021, and March 2022. The dominant attack vectors are SQL Injection (SQLI) and Local File Inclusion (LFI), represented by blue and red bars, respectively, showing a substantial rise in activity over the period. Other vectors such as Command Injection (CMDI), PHP Injection (PHPI), JavaScript Injection (JSI), Remote File Inclusion (RFI), and Malformed URL (MFU) are present but less prominent. This trend underscores the growing threat landscape for web applications and APIs, highlighting the need for enhanced security measures.

The integration of Nginx and ModSecurity with threat intelligence presents a powerful and efficient solution for securing web applications against a wide range of cyber threats. This integration not only demonstrates the practical application of these technologies but also highlights the importance of proactive and dynamic security measures in the ever-evolving landscape of cybersecurity.

1.3 Existing System

The existing system comprises a hybrid web application firewall designed to enhance the security of web applications by integrating both signature-based detection and anomaly detection methods. This approach aims to mitigate the weaknesses of each individual method, ensuring a more robust defense mechanism against web-based attacks.

The hybrid system utilizes signature-based detection to quickly identify and block known attack patterns using a blacklist of attack signatures. While this method is efficient and effective for known attack types, it struggles to detect zero-day attacks, which are previously unknown vulnerabilities.

To address this limitation, the proposed system incorporates anomaly detection, which identifies unusual or suspicious behavior in HTTP requests that deviate from the norm. Anomaly detection relies on statistical methods to establish what constitutes normal behavior for a web application and uses this baseline to identify anomalies. Key parameters for anomaly detection include the length of the request, the frequency of specific characters within the request, and the overall number of requests.

The system works by analyzing HTTP requests before they reach the web server. Requests are first processed through the signature-based detection module, and any matching known attack signatures result in the request being blocked. Requests that pass this initial check are then subjected to anomaly detection, which evaluates their characteristics against the established normal behavior metrics.

Specifically, the anomaly detection process involves several key steps:

1. **Analysis of Request Count:** Normal requests tend to repeat frequently, whereas attack requests are less likely to repeat. Requests that repeat more than a certain threshold (e.g., 15 times) are considered normal.
2. **Analysis of Request Length:** The length of requests is compared against typical values. Requests significantly longer or shorter than the average may indicate an attack.
3. **Analysis of Request Frequency**: The frequency of characters in the request is analyzed. Requests with character frequencies that deviate significantly from the norm are flagged as anomalies.

By combining these two detection methods, the hybrid firewall aims to provide comprehensive protection against both known and unknown attacks, offering a more secure environment for web applications.

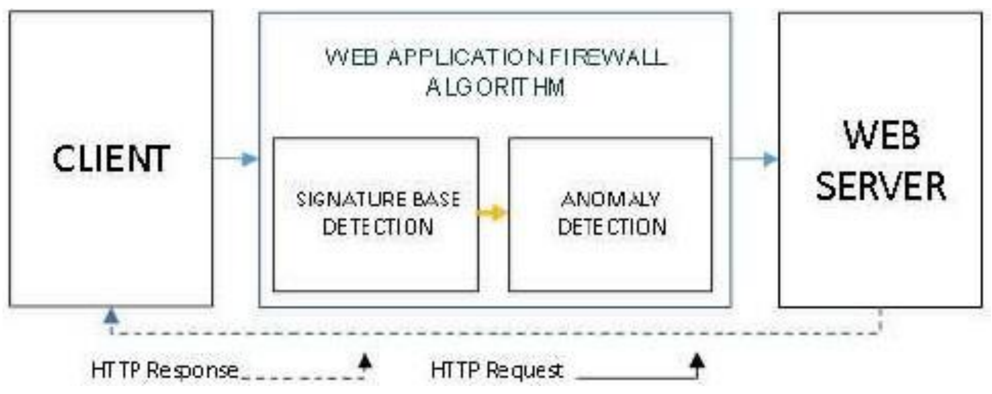


Figure 1.3: Block diagram of the hybrid firewall model

The existing system includes a detailed design with a block diagram (figure 1.3) that outlines the flow of HTTP request processing through the signature-based and anomaly detection modules. This design ensures that all incoming requests are thoroughly examined before being allowed to interact with the web server, thereby reducing the risk of successful attack.

The hybrid web application firewall leverages the strengths of both signature-based and anomaly detection methods to provide a robust defense mechanism against a wide range of web-based attacks, enhancing the overall security of web applications

1.3.1 Demerits of an Existing System

The existing system is a hybrid web application firewall that combines signature-based detection and anomaly detection to prevent web-based attacks. While this hybrid approach aims to leverage the strengths of both detection methods, it also has several demerits:

1. **Performance Overhead**: The implementation of both signature-based and anomaly detection methods can lead to significant performance overhead. Signature-based detection requires constant updating of signature databases, while anomaly detection involves complex statistical analysis. This can slow down the system, especially under high traffic conditions.

2. **Limited Zero-Day Attack Detection**: Although anomaly detection can potentially identify zero-day attacks by recognizing deviations from normal behavior, it is not reliable. The system relies on predefined statistical models, which might not capture all possible attack vectors. Consequently, sophisticated attackers can craft payloads that mimic normal traffic, bypassing detection.

3. **High False Positives and Negatives**: The hybrid approach may result in high rates of false positives and negatives. False positives occur when legitimate requests are mistakenly identified as attacks, disrupting normal operations and frustrating users. False negatives, on the other hand, happen when actual attacks go undetected, leaving the system vulnerable.

4. **Complex Configuration and Maintenance**: Setting up and maintaining a hybrid detection system is complex and requires specialized knowledge. Regular updates to the signature database, fine-tuning of anomaly detection models, and continuous monitoring are necessary to ensure effectiveness. This increases the administrative burden and the likelihood of configuration errors.

5. **Scalability Issues**: As web traffic grows, the system may struggle to scale effectively. The computational resources needed for real-time analysis of HTTP requests can become a bottleneck. Ensuring scalability while maintaining detection accuracy is a significant challenge.

6. **Dependency on Training Data**: Anomaly detection models require extensive training data to accurately distinguish between normal and malicious behavior. If the training data is not comprehensive or up-to-date, the models might produce inaccurate results. Additionally, the system needs to adapt to changes in normal traffic patterns over time, which requires continuous retraining.

7. **Resource Intensive**: The existing system demands considerable computational resources for processing and analyzing HTTP requests. This includes memory, CPU, and storage for maintaining logs and running statistical analyses. Organizations with limited resources might find it difficult to implement and sustain such a system.

1.4 Proposed System

To enhance the effectiveness of web application security, the firewall is integrated with threat intelligence feeds. By integrating threat intelligence feeds, the firewall gains the capability to significantly reduce false positives through the correlation of incoming traffic against known malicious indicators. This approach enhances the accuracy of threat detection by ensuring that legitimate user traffic is not erroneously blocked. Leveraging real-time threat intelligence data enables the firewall to stay updated with the latest information on malicious activities targeting web applications. This proactive stance not only enhances the effectiveness of web application security but also strengthens the overall defense posture against evolving cyber threats.

Threat intelligence feeds play a crucial role in providing the firewall with current insights into known malicious actors, attack signatures, indicators of compromise (IoCs), and emerging cyber threats. By continuously updating its rule sets based on this intelligence, the firewall adapts dynamically to new threats. This adaptive capability allows the firewall to incorporate new signatures and IoCs into its detection mechanisms promptly, thereby enhancing its ability to detect and prevent malicious activities in real-time.

To ensure the seamless integration of threat intelligence data, the firewall employs event-driven triggers or scheduled tasks. These mechanisms automate the process of updating rule sets based on the latest threat intelligence feeds. By automating this process, the firewall reduces the dependency on manual intervention and ensures that it remains responsive to emerging threats without delay. This proactive approach not only strengthens the security posture of web applications but also mitigates risks more effectively by staying ahead of potential threats.

The component diagram in figure 1.4 outlines the architecture of a Web Application Firewall (WAF) system designed to secure and manage a web application. At the forefront is **Nginx with ModSecurity**, positioned on the left, serving as the front-end security layer. It filters incoming traffic and blocks malicious requests, ensuring that only legitimate traffic reaches the web application.

In the center of the diagram is the **Application Server**, which handles backend processing. It receives secure requests from Nginx with ModSecurity and processes them according to the application's business logic. To the right of the Application Server is the **Database** component, responsible for managing data storage and retrieval, supporting the application's data needs.

Below the Application Server, the **Log and Monitoring System** oversees system health and security by collecting logs from both Nginx with ModSecurity (security events) and the Application Server (system events). This centralized logging allows for effective monitoring and management of the system's performance and security.

The system also includes a **User** actor on the far left, representing the end-user who accesses the web application through the secured Nginx with ModSecurity. This architecture combines security, processing, data management, and monitoring to ensure the web application operates securely and efficiently.

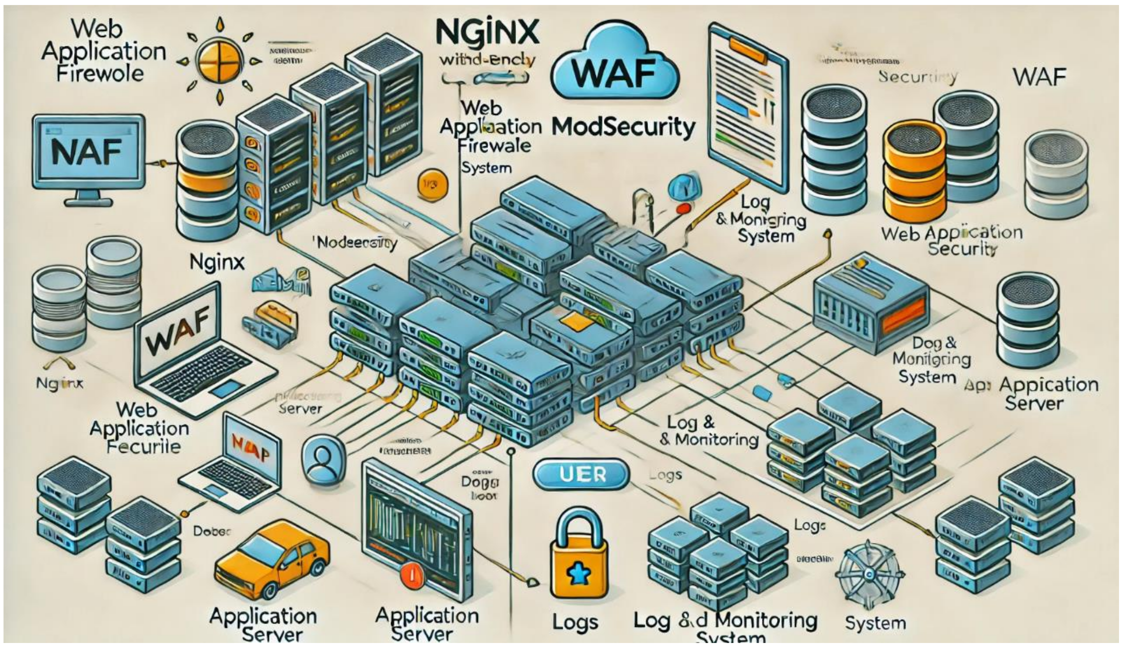


Figure 1.4: Component Diagram

1.4.1 Merits of Proposed System

The proposed system offers several significant enhancements in web application security by integrating Nginx with ModSecurity:

1. **Robust Protection Against Cyber Threats**: The system effectively filters and blocks malicious requests, significantly reducing the risk of common attacks such as SQL

injection, cross-site scripting (XSS), and other exploitations that can compromise web application integrity and availability.

2. **High Performance and Flexibility**: Nginx is known for its efficiency, scalability, and low resource consumption. By combining Nginx with ModSecurity, the system maintains optimal performance, ensuring smooth processing of legitimate user traffic while providing robust security.

3. **Virtualized Testing Environment**: Using a virtual machine with Parrot OS allows for testing in a controlled and isolated environment, minimizing risks to the production environment. This setup enables rigorous testing and validation of the WAF, ensuring its effectiveness without impacting live operations.

4. **Dynamic and Customizable Security Rules**: The system develops custom security rules derived from threat intelligence feeds, ensuring it stays up-to-date with the latest threats and can adapt to new attack patterns. Automating the extraction and conversion of these rules into a ModSecurity -compatible format ensures the firewall is always equipped with relevant security measures.

5. **Comprehensive Documentation and Reporting**: Detailed documentation of each step in the installation, configuration, and testing processes provides a valuable resource for future maintenance and enhancements. Reports on the system's performance and simulated attack results offer insights into its effectiveness and highlight areas for improvement.

Overall, the proposed system enhances web application security by providing robust protection, maintaining high performance, utilizing a virtualized testing environment, employing dynamic and customizable security rules, and offering comprehensive documentation and reporting. This thorough approach ensures the system's effectiveness and facilitates ongoing security efforts.

1.5 Problem Statement

1. **Limited Effectiveness Against Emerging Attacks:** Traditional web application firewalls (WAFs) struggle to keep pace with rapidly evolving cyber threats. They

typically rely on static rule sets that are effective against known vulnerabilities but often fail to detect and mitigate emerging or zero-day attacks. This limitation arises because static rules cannot adapt quickly to new attack techniques and evasion methods used by cybercriminals.

As a result, web applications remain vulnerable to sophisticated threats that exploit unknown weaknesses, potentially leading to data breaches, service disruptions, and unauthorized access. Addressing this challenge requires a more dynamic approach that integrates real-time threat intelligence to continuously update WAF rules and proactively defend against emerging threats.

2. **Minimize False Positives:** False positives pose a significant operational challenge for web application security. These occur when legitimate user activities are incorrectly flagged as malicious and blocked by the WAF. The issue stems from the overly cautious nature of static rule sets, which may interpret benign anomalies or unusual but legitimate behaviors as security threats. False positives not only disrupt normal business operations and degrade user experience but also consume valuable resources as security teams must manually investigate and resolve flagged incidents.

   To mitigate this challenge, advanced WAF solutions employ intelligent algorithms and machine learning techniques to analyze traffic patterns and accurately distinguish between genuine threats and harmless anomalies. Integrating real-time threat intelligence further enhances the WAF’s ability to minimize false positives by providing contextual awareness of current cyber threats and attack trends.

1.6 Objectives

1. To address the challenges of limited effectiveness against emerging attacks, the objectives focus on enhancing the web application firewall (WAF)'s capabilities to proactively defend against evolving cyber threats. This includes developing advanced threat detection mechanisms to identify and mitigate evolving threats swiftly.

Integrating real-time threat intelligence feeds into the WAF infrastructure aims to continuously update and refine rule sets based on the latest global threat information, ensuring the WAF remains agile and responsive. Additionally, implementing dynamic rule management allows the WAF to adapt quickly to new attack vectors without manual intervention, bolstering its ability to preemptively block emerging threats and strengthen overall cybersecurity posture.

2. Simultaneously, the objectives aim to minimize false positives by implementing advanced traffic analysis techniques using intelligent algorithms. These methods enable the WAF to accurately distinguish between legitimate user activities and genuine security threats, thereby reducing disruptions caused by false alarms.

Enhancing behavioral analysis and contextual awareness further improves the WAF's ability to understand normal application behaviors and adapt its security measures accordingly, optimizing rule set tuning to effectively filter malicious traffic while maintaining seamless user experience. By achieving these objectives, the aim is to enhance the overall effectiveness of the WAF in safeguarding web applications against sophisticated cyber threats.

Chapter 2
Literature Survey

[1] Adem et. Al, In their exploration titled "Development of a Hybrid Web Application Firewall to Prevent Web Based Attacks" by Adem Tekerek, Cemal Gemci, and Omer Faruk Bay discusses the creation of a hybrid web application firewall to address vulnerabilities in web applications that traditional network-layer firewalls cannot handle. Adem Tekerek, Cemal Gemci, and Omer Faruk Bay developed a hybrid web application firewall to address web application vulnerabilities that traditional firewalls can't handle. Recognizing that about 70% of web-based attacks succeed, the proposed system combines signature-based and anomaly detection methods to filter HTTP requests.

Initially, signature-based detection identifies known attack patterns. If no match is found, anomaly detection analyzes request count, length, and letter frequency to detect deviations from normal behavior. Anomaly detection establishes a baseline of normal behavior, blocking requests that exceed specific thresholds or exhibit unusual character frequencies. Signature-based detection involves maintaining an updated blacklist of known attack signatures. Results demonstrate that the hybrid approach enhances the accuracy of identifying and preventing attacks, effectively distinguishing between normal and malicious requests. Integrating both detection methods significantly improves web application security, providing a robust mechanism to counter various web-based threats.

[2] Firkhan Ali Bin Hamid Ali in “A Study of Technology in Firewall System” provides an overview of firewall technologies, detailing their methods, advantages, and disadvantages. Firewalls protect networks from unauthorized access by regulating traffic through routers connecting to network infrastructures. Both hardware and software firewalls play crucial roles in network security. Packet filtering checks each network packet against predefinedrules to allow or block them, utilizing network interface cards to connect internal and external networks. This method is simple, offers low-level control, identifies spoofed packets, and serves as a central access control point, often integrated into routers. However, it involves complex rule setup, vulnerable open ports, and doesn't cover non- network-based attacks, making it best suited for smaller networks with lower security needs.

Dynamic packet inspection tracks network connections and the state of packets to make informed decisions about traffic, enhancing security by blocking unsolicited incoming packets. Although it requires a more complicated setup, it offers flexibility and sophistication, including additional services like redirecting connections toauthentication services and managing web traffic. Application proxies act as intermediaries for client requests, providing detailed control over traffic at the applicationlevel. They offer high security and control by inspecting entire packet contents but are more resource-intensive and can introduce latency. Each firewall technology has its strengths and weaknesses, making them suitable for different scenarios and security needs. Packet filtering is simpler but less secure, dynamic packet inspection offers bettercontrol and security at the cost of complexity, and application proxies provide the highestsecurity but require more resources.

[3] Lin Zhang and Mengxing Huang in “A Firewall Rules Optimized Model Based On Service-Grouping” propose an optimized model for firewall rules aimed at reducing the number of rules and the time required for rule filtering while maintaining firewall performance.

The model merges firewall policy rules based on rule-service, detects rules quickly using a rule-service-based algorithm, resolves conflicts with an action constraint strategy, and merges rules without anomalies. As network sizes grow, the number of filtering rules increases, leading to slower filtering processes and potential performance overloads. Previous research has focused on improving the efficiency of rule anomaly detection algorithms and rule filtering using various techniques. The proposed model addresses these challenges by dividing rules into service groups, eliminating anomalies within each group, and running a merging algorithm.

Anomalies are detected and resolved within service groups using action constraints, and the firewall rule set is updated after merging. The policy rules merging algorithm optimizes the firewall by grouping rules according to services and merging rules within each group if their IP domains are continuous. The experimental results show that the optimized model outperforms other rule merging models in terms of merging efficiency and reduces the number of filtering hits during packet processing. This service-grouping-based model effectively optimizes firewall rules, reducing processing time and the number of rules while maintaining security and performance. The novel approach enhances firewall performance by focusing on service-grouping and efficient rule merging.

[4] Z. Ghanbari, Y. Rahmani, M. Hossein Ahmadzadegan's "A Comparative Approach for Web Application Firewall Using Penetration Testing Security Assessment" discusses the effectiveness of different web application firewalls (WAFs) in enhancing web application security. Emphasizing the growing need for robust security measures due to the rise in web-based attacks, the research highlights both signature-based and anomaly detection methods are used to develop a hybrid WAF model, combining the strengths of both approaches. Findings indicate that while signature-based detection is faster, it is less effective against zero-day attacks, whereas anomaly detection, though more effective against such attacks, can be slower and resource-intensive.

The hybrid approach leverages the speed of signature-based detection with the comprehensive protection offered by anomaly detection. Conclusions indicate that the hybrid WAF model provides a more robust solution to web application security by addressing the limitations of both detection methods. Continuous monitoring and assessment are emphasized as crucial for maintaining WAF effectiveness and adapting to new threats. The study underscores the importance of using a combination of detection methods to enhance web application security, providing a comprehensive evaluation of WAFs through penetration testing and security assessment.

Chapter 3
System Design

To achieve the objectives of enhancing web application security and minimizing false positives, the system is designed by integrating ModSecurity with a robust threat intelligence framework. The system architecture begins with deploying ModSecurity as a web application firewall (WAF) on critical servers, where it operates as an intermediary between users and web applications. This setup enables real-time inspection of incoming and outgoing traffic, leveraging ModSecurity's extensive rule sets to detect and block malicious activities such as SQL injection, cross-site scripting (XSS), and other web-based threats. To enhance its detection capabilities, threat intelligence feeds are incorporated that provide up-to-date information on emerging threats, enabling ModSecurity to dynamically adjust its rules and respond proactively to new attack vectors.

Furthermore, the system design includes integrating ModSecurity with existing security tools and protocols to ensure comprehensive protection. This involves setting up logging and monitoring mechanisms that feed data into a centralized Security Information and Event Management (SIEM) system. By correlating ModSecurity alerts with data from other security tools, a holistic view of the threat landscape and streamline incident response can be achieved. This comprehensive approach to system design not only fortifies web application defenses but also aligns with the objectives of reducing security incidents and maintaining high operational efficiency.

3.1 System Architecture

The system architecture diagram in figure 3.1 that represents the interaction and data flow between various components in a secure web application environment. The diagram illustrates how different elements work together to process a user's request while ensuring security through multiple layers of inspection, logging, and threat intelligence integration.

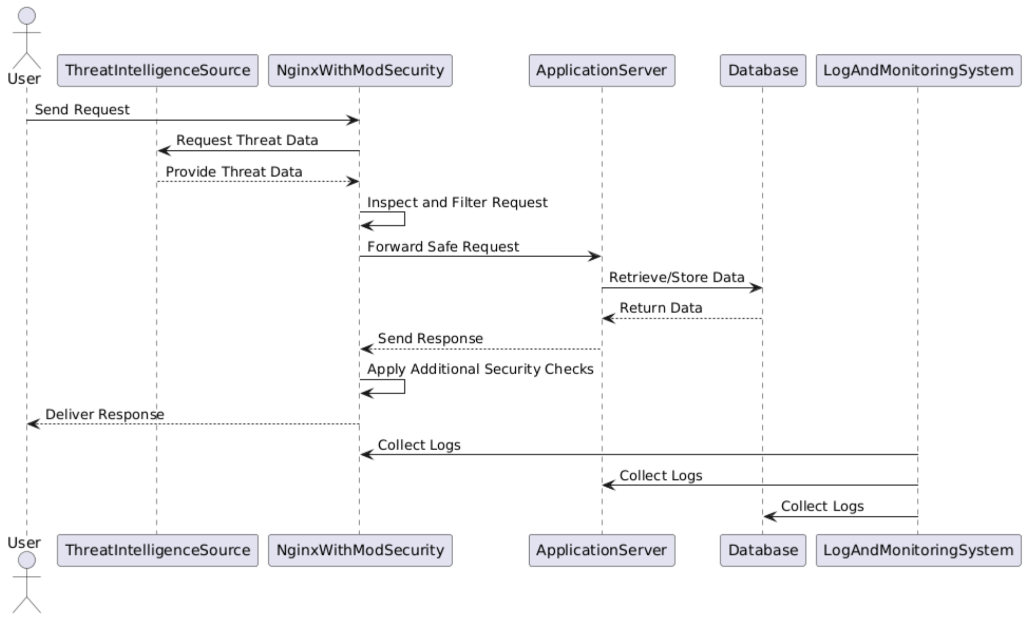


Figure 3.1: System Architecture

At the start, the user sends a request that first reaches the **Nginx with ModSecurity** component, which acts as a reverse proxy server with integrated web application firewall capabilities. Before processing the request, Nginx with ModSecurity queries a **Threat Intelligence Source** to obtain up-to-date threat data. This data is used to inspect and filter the incoming request, ensuring it is safe to proceed. Once the request passes these security checks, it is forwarded to the **Application Server**.

The Application Server, which handles the core logic of the application, then interacts with the **Database** to either retrieve or store data based on the user's request. After processing, the data is returned to the Application Server. Before sending the response back to the user, the Application Server applies additional security checks to ensure the integrity and security of the outgoing data.

Throughout this process, logging is an integral part of the system's security and monitoring strategy. Logs are collected at various stages, including by Nginx with ModSecurity when inspecting the request, by the Application Server during data processing, and by the **Log and Monitoring System**, which aggregates logs from the Application Server and Database. This comprehensive logging mechanism helps in monitoring, auditing, and responding to potential security incidents.

Finally, the response is delivered back to the user, completing the cycle. The diagram underscores the importance of integrating security at multiple points in the data flow, from the initial request inspection to the final response, with continuous logging and monitoring to ensure a secure and robust application architecture.

3.2 Data Flow Diagram

The DFD diagram (figure 3.2) depicts an architecture designed to handle user requests securely and handle data efficiently.

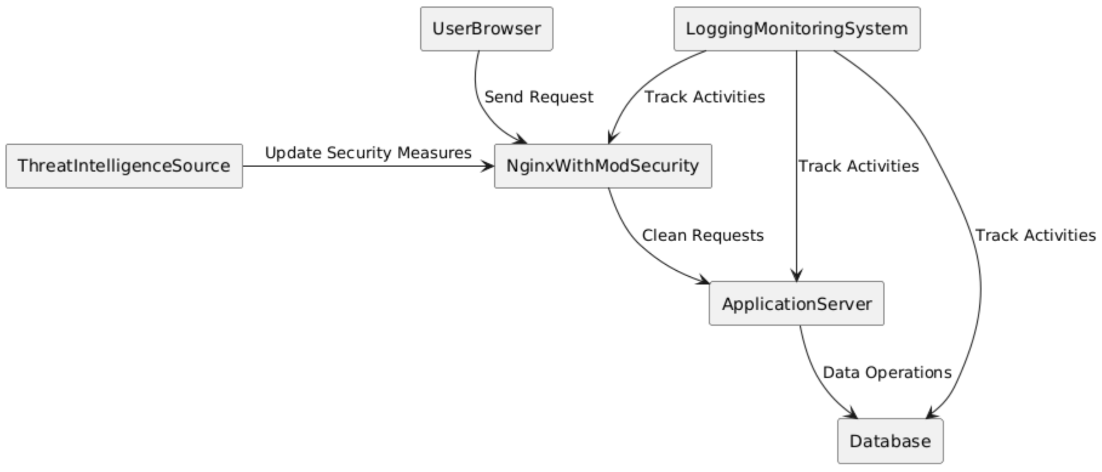


Figure 3.2: Data Flow Diagram

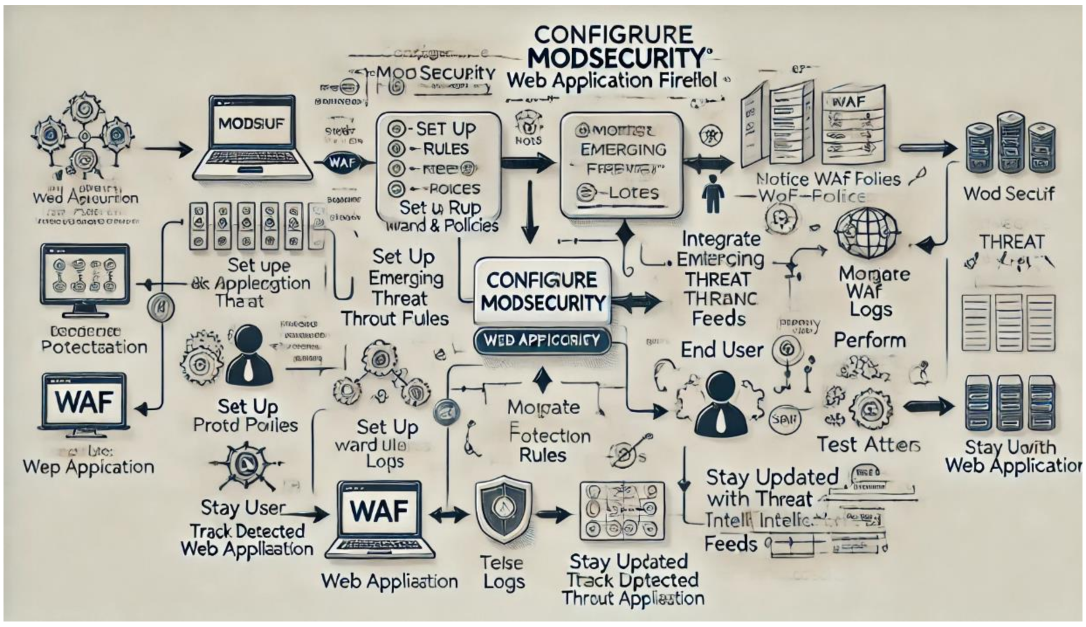
The Data Flow Diagram (DFD) showcases the secure data processing workflow within a web application environment. The process begins with a user sending a request via the **UserBrowser**, which is received by **Nginx with ModSecurity**. This component cleans and secures the request using the latest threat data provided by a **Threat Intelligence Source**. Once the request is verified, it is passed to the **Application Server**, where the core processing takes place, including interactions with the **Database** for retrieving or updating data.

Simultaneously, a **Logging and Monitoring System** tracks all activities across the key components—Nginx with ModSecurity, Application Server, and Database—to ensure real-time security monitoring and auditing. This system helps detect any unusual or malicious behavior, maintaining the integrity and security of the entire application. The diagram underscores the importance of integrating threat intelligence and continuous monitoring throughout the data processing pipeline.

3.3 UML Diagrams

Unified Modeling Language (UML) is a standardized visual modeling language crucial in object- oriented software engineering. Developed and maintained by the Object Management Group, UML aims to provide a universal language for modeling object-oriented software systems. It consists of a meta-model defining its structure and a notation for graphical representation. UML's primary purpose is to offer a comprehensive tool for specifying, visualizing, constructing, and documenting software systems and business processes. It encapsulates best practices in engineering that have proven effective in modeling complex systems. By using predominantly graphical notations, UML makes it easier for developers to express and communicate software designs.

The language is designed with several key goals in mind. It strives to be expressive and user- friendly, allowing for the creation and exchange of meaningful models. UML offers extensibility mechanisms to adapt to specific needs while remaining independent of particular programming languages or development processes.



CONFIGRURE MODSECURITY
MOD Security Web Application Firellol
MODSUF
WAF
SET UP RULES
POICES
Set u Rup Ward & Policies
MONITOR EMERGING
LOTES
Notice WAF Folies
WAF
Wod Secuif
THREAT
Wed Aprcurtion
Set upe Applecgtion Thaat
Set Up Emerging Threat Throut Fules
CONFIGURE MODSECURITY
WED APPICORICY
Integrate Emercing THREAT THRANC Feeds
Mongate WAF Logs
Perform
Potectation
End User
WAF
Wep Appication
Set Up Protd Poiles
Set Up ward Ule Lops
Mongate Fotection Rules
Test Atten
Stay Uoith Web Application
Stay Updated with Threat Intelli Intelic Feeds
Stay User Track Detected Web Appliation
WAF
Web Appliation
Telse Logs
Stay Updated Track Dptected Throut Appliation

Figure 3.3: UML Diagram

It provides a formal basis for understanding modeling concepts and supports higher-level development paradigms such as collaborations, frameworks and design pattern

The UML diagram in figure 3.3 outlines a comprehensive workflow for configuring ModSecurity, a Web Application Firewall (WAF). The process begins with the "Configure ModSecurity" step, where an administrator initiates the setup of WAF configurations. The next step, "Set Up Rules and Policies," involves defining specific protection rules to filter and manage web traffic. This is followed by "Monitor WAF Logs," where the system continuously tracks detected threats, providing critical insights into potential security incidents. Each stage is connected by directional arrows, clearly illustrating the sequence of operations and the logical flow of decision-making required to secure a web application effectively.

The workflow then advances to more dynamic security measures. "Integrate Emerging Threat Rules" is focused on regularly updating the WAF with new threat rules to address evolving security challenges. "Stay Updated with Threat Intelligence Feeds" ensures that the firewall is equipped with the latest threat data, enhancing its defense capabilities. "Perform Test Attacks" follows, where simulated threats are used to validate and stress-test the WAF's configuration and effectiveness. The final stage, "End User Accesses Web Application," signifies that users interact with a web application protected by a robust and well-maintained WAF, ensuring comprehensive security and reliable access. The UML diagram effectively conveys the critical stages and continuous improvement necessary for optimal web application security management.

3.3.1 Use case Diagram

One of the key strengths of use case diagrams lies in their ability to bridge the gap between technical and non-technical stakeholders. They serve as a universal language, enabling effective communication about system requirements and functionalities across diverse project teams. This shared understanding is crucial for aligning expectations, identifying potential issues early in the development process, and ensuring that the final system meets user needs.

The Use Case diagram in figure 3.4 depicts the process of managing a Web Application Firewall (WAF). It identifies the "Administrator" as the primary actor responsible for initiating critical actions like "Configure ModSecurity" and "Set Up Rules and Policies." These foundational steps are essential for establishing the WAF's initial configuration and defining the rules that filter and manage incoming web traffic, ensuring the web application's security.

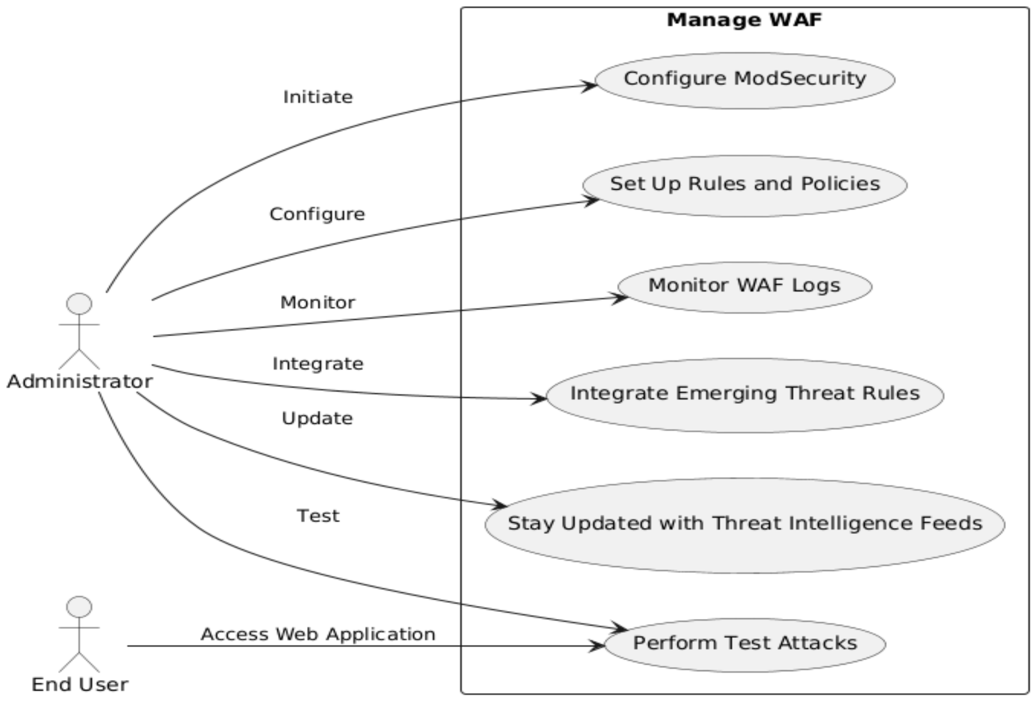


Figure 3.4: Use-Case Diagram

The diagram further outlines the Administrator's role in ongoing WAF management, such as "Monitor WAF Logs," "Integrate Emerging Threat Rules," and "Stay Updated with Threat Intelligence Feeds." These activities are crucial for adapting the WAF to evolving threats by continuously updating threat rules and integrating new intelligence feeds. This proactive approach ensures that the WAF remains effective against new vulnerabilities and sophisticated attack vectors.

Lastly, the diagram includes the "Perform Test Attacks" action, highlighting the need for regular testing to assess the WAF's effectiveness and resilience. This step ensures that the WAF can protect the web application, providing secure access for end users. The UML Use Case diagram effectively illustrates the comprehensive responsibilities of an Administrator in maintaining a robust WAF environment, emphasizing continuous improvement and adaptation to the threat landscape.

Use case diagrams play a vital role in the iterative process of system design, allowing for continuous refinement and validation of system requirements as the project progresses. By providing a clear, visual representation of system behavior, these diagrams help in identifying gaps, redundancies, or inconsistencies in the planned functionality, thereby contributing to more robust and user- centric system designs.

3.3.2 Class Diagram:

The diagram (figure 3.5) illustrates a relational database schema involving three key entities: User, Request, and Response. Each entity is represented as a table with specific attributes.
The User entity represents system users and includes attributes such as userID (a unique identifier for each user) and userName (the name of the user). Users are linked to the Request entity, indicating that users can initiate requests within the system.

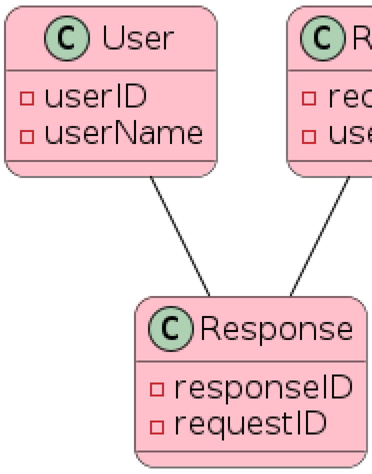


Figure 3.5: Class Diagram

The Request entity represents individual requests made by users and includes attributes like requestID (a unique identifier for each request) and userID (identifying the user who made the request). This establishes a relationship between users and their corresponding requests.

The Response entity captures the responses generated for each request, with attributesincluding responseID (a unique identifier for each response) and requestID (indicating the request to which the response corresponds). This relationship links each response back to its originating request.

Overall, the schema outlines a structured approach where users interact with the system by making requests, each of which may result in one or more responses. The relationships between User, Request, and Response entities facilitate tracking and managing user interactions and their outcomes within the database.

3.3.3 Sequence Diagram:

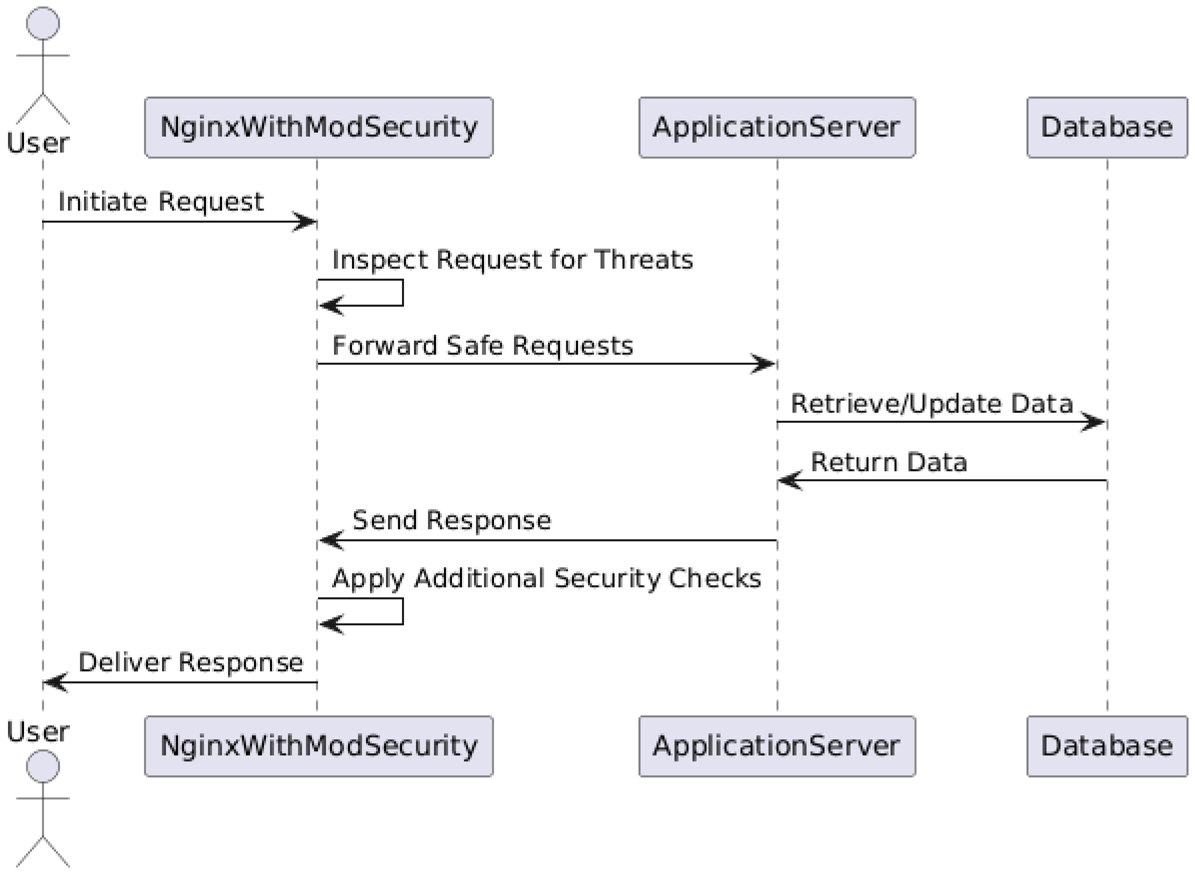


Figure 3.6: Sequence Diagram

This sequence diagram in figure 3.6 illustrates a secure interaction flow between a user, a security-enhanced Nginx web server (with ModSecurity), an application server, and a database. The process begins with the user's request being inspected by Nginx with ModSecurity for potential threats. If the request is deemed safe, it is forwarded to the application server, which processes the request, often involving interaction with the database to retrieve or update data.

After the database returns the necessary data, the application server sends the response back to Nginx with ModSecurity. Before delivering the response to the user, Nginx applies additional security checks to ensure the response is secure. The diagram highlights a multi-layered security approach, emphasizing that both incoming requests and outgoing responses are carefully monitored and filtered to maintain a secure and reliable interaction.

3.3.4 Activity Diagram:

The activity diagram in figure 3.7 represents the process of handling a user request within a web application system, focusing on security and data retrieval. The process begins with a user request, which is immediately inspected by Nginx integrated with ModSecurity. During this inspection phase, the request is analyzed for any malicious content. If the request is identified as malicious, it is blocked, preventing it from proceeding further into the system.

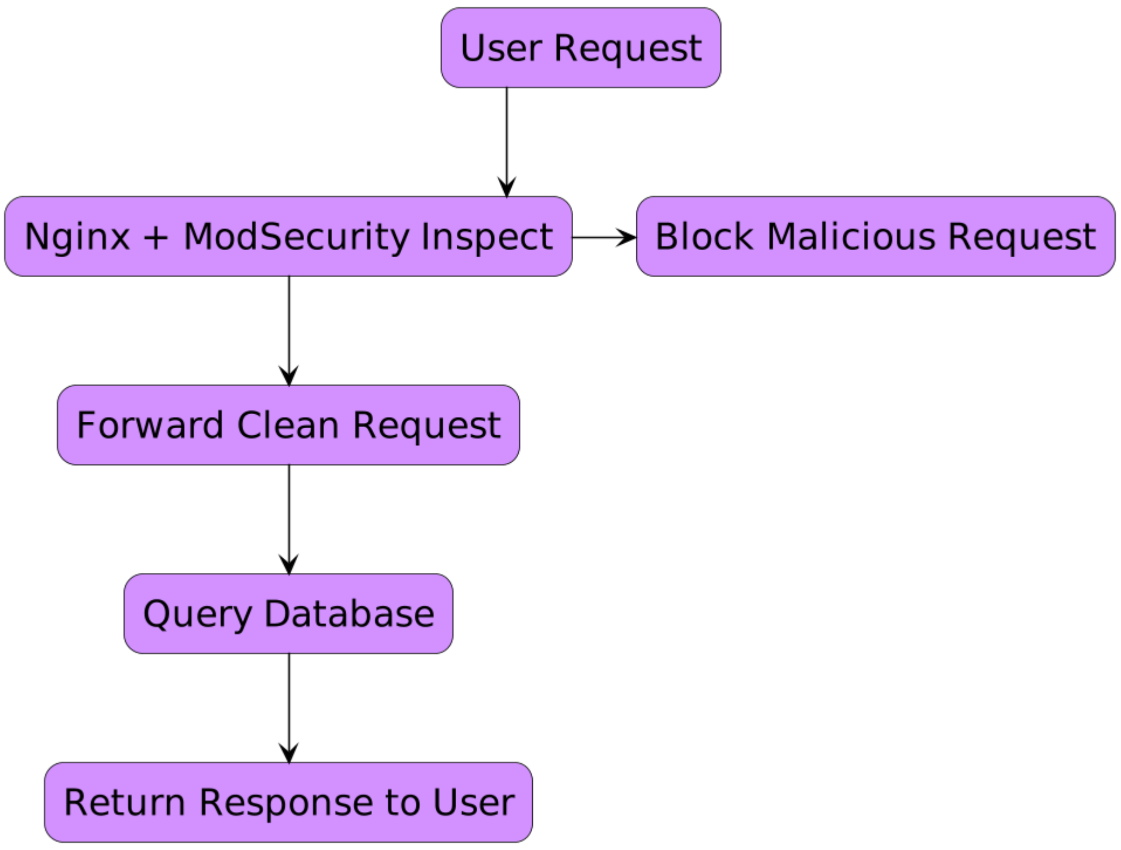


Figure 3.7: Activity Diagram

If the request passes the security inspection and is deemed clean, it is forwarded to the next step, where the system queries the database to retrieve or update data as needed. Once the database operation is complete, the system returns the processed response to the user. This diagram highlights a streamlined approach to handling requests, ensuring that only safe requests are processed and responded to, thereby maintaining the integrity and security of the system.

Overall, the architecture ensures that only safe and legitimate user requests are processed, enhancing system security and maintaining efficient data handling throughout the interaction cycle in web applications. Ultimately, this process contributes to a more secure and reliable user experience, while safeguarding sensitive data from potential threats.

3.4 Requirements

3.4.1 Hardware Requirements

Hardware specifications are integral to the seamless deployment and optimal performance of software systems.

CPU: Multi-core processors (e.g., Intel Xeon or AMD Ryzen)

Multi-core processors suchas Intel Xeon or AMD Ryzen are essential for handling concurrent requests and processing intensive rulesets used by ModSecurity. These processors offer the computing power needed to efficiently inspect and filter web traffic in real-time, ensuring minimal impact on applicationperformance.

RAM: Minimum 8GB RAM, recommended 16GB or more

Adequate RAM is critical for storing and processing application data and rulesets efficiently. With a minimum of 8GB RAM(preferably 16GB or more), the server can handle simultaneous connections and maintain optimal performance during peak traffic periods, preventing bottlenecks that could degrade user experience or compromise security.

Storage: SSD storage

SSD storage is preferred over traditional HDDs due to its faster read/write speeds, which are crucial for accessing and managing ModSecurity logs, configuration files, and database entries. This speed ensures that the WAF can process and respond to security events promptly, minimizing response times during incident detection andresolution.

Bandwidth: High-speed internet connection or LAN

A high-speed internet connection or LAN with sufficient bandwidth is essential for handling incoming and outgoing web traffic efficiently. This bandwidth ensures that the WAF can inspect and filter web requests and responses without introducing latency or affecting application performance, thereby maintaining seamless user experiences.

Network Interface Cards (NICs): Gigabit Ethernet interfaces

Gigabit Ethernet interfacesenable fast data transmission between servers and clients, supporting the rapid exchange of web traffic data necessary for real-time threat detection and mitigation. These NICs reduce latency and ensure that the WAF can handle high volumes of network traffic effectively, enhancing overall security posture.

Backup Solution: Automated backup system

An automated backup system is critical for ensuring data integrity and operational continuity.This system should regularly back up ModSecurity configuration files, logs, and any associated databases. Automated backups help mitigate the risk of data loss due to hardware failures, accidental deletions, or security incidents, enabling quick recovery and minimal downtime.

Redundancy: Failover mechanisms

Implementing failover mechanisms ensures continuous availability of ModSecurity and associated services. Redundancy can involve deploying multiple servers in a load-balanced configuration or using high-availability clustering. These measures prevent single points of failure, ensuring that the WAF remains operational even if one server or component fails, thereby maintaining uninterrupted protection against web-based threats.

Cooling and Ventilation: Adequate cooling systems

Adequate cooling systems are essentialto maintain optimal operating temperatures for server hardware. Proper cooling prevents overheating, which can lead to hardware malfunctions or failures. Maintaining a stable temperature in server rooms or data centers ensures the longevity and reliability of critical infrastructure components.

Power Supply: Uninterruptible Power Supply (UPS)

An Uninterruptible Power Supply (UPS) is crucial to prevent data loss and system downtime during power outages or fluctuations. UPS systems provide temporary power backup, allowing servers and network equipment to remain operational long enough for graceful shutdowns or continued operation until normal power is restored. This safeguards against data corruption and ensures continuousoperation of the WAF and associated security services.

In short, these hardware requirements form the foundation for deploying a resilient and high-performance hybrid web application firewall integrated with threat intelligence feeds and a SIEM system. By investing in robust server hardware, reliable network infrastructure, backup solutions, redundancy measures, and environmental controls, organizations can effectively enhance their cybersecurity posture, ensuring proactive threat detection, rapid response capabilities, and continuous protection against evolving web-based threats.

• **System**	:	High-performance server with multi-core processors (e.g., Intel Xeon or AMD Ryzen)
• **Hard Disk**	:	Solid State Drive (SSD) with at least 500GB

		capacity
• **Monitor**	:	High-resolution monitor
• **Mouse**	:	Standard optical mouse
• **RAM**	:	Minimum 16GB RAM
• **Network Interface Cards (NICs)**	:	Gigabit Ethernet interfaces
• **Power Supply**	:	Uninterruptible Power Supply (UPS)
• **Cooling System**	:	Robust cooling solutions

3.4.2 Software Requirements

Software requirements serve as a comprehensive outline detailing the essential components and dependencies necessary for the effective operation of a software application. These specifications encompass critical aspects such as the required operating system, programming language, development tools, design technologies, and database management systems. Each component is carefully chosen to ensure compatibility, functionality, and efficiency in executing the software's intended tasks and functionalities.

ModSecurity Version: Latest stable release (e.g., ModSecurity 3.x)

Using the latest stablerelease of ModSecurity ensures that the WAF leverages the most recent security enhancements, bug fixes, and performance optimizations. Newer versions often introduce improved detection capabilities for evolving web-based attacks.

Source: Reliable threat intelligence providers or open-source threat feeds

Accessing reliable threat intelligence sources or open-source feeds ensures that the WAF is equipped with up-to-date information about known threats and attack patterns. This information is crucial for making informed decisions about threat detection and response.

Platform: Compatible with the organization’s chosen SIEM solution (e.g., Splunk,Elastic SIEM)

Choosing a SIEM system compatible with ModSecurity allows for centralized monitoring, analysis, and correlation of security events across the organization’s IT infrastructure. This integration streamlines incident response and enhances overall security posture.

Linux Distribution: Preferred distributions include Ubuntu Server, CentOS, or Debian

Using a Linux distribution such as Ubuntu Server, CentOS, or Debian provides a stable and secure operating environment for deploying ModSecurity and other security components. These distributions offer robust support for server applications and security updates.

Database: MySQL or PostgreSQL for storing ModSecurity logs and configuration data
Utilizing a relational database management system (RDBMS) like MySQL or PostgreSQL allows efficient storage and management of ModSecurity logs and configuration data. This capability supports forensic analysis, compliance reporting, and ongoing optimization of WAF performance.

• **Operating System**	:	Parrot OS
• **Coding Language**	:	Python, Linux
• **Front-end**	:	HTML, CSS
• **Designing**	:	HTML, CSS, JavaScript
• **Database**	:	MySQL
• **Server**	:	Nginx
• **Firewall**	:	ModSecurity
• **Threat Intelligence Feed**	:	Alien Vault OTX

In conclusion, each of these software requirements plays a vital role in the development and operation of a hybrid web application firewall integrated with threat intelligence feeds and a SIEM system. Together, they form a comprehensive security solution that enhances the detection, prevention, and response capabilities against web-based attacks, therebysafeguarding critical assets and ensuring the resilience of web applications in dynamic threat environments.

3.5 System Study

The system study phase initiates with a thorough evaluation of the existing web application security infrastructure within the organization. This includes assessing the efficacy of current firewalls, security protocols, and incident response mechanisms in mitigating prevalent web-based threats such as SQL injection and cross-site scripting (XSS). By scrutinizing these components, the goal is to identify strengths and weaknesses in the organization'ssecurity posture, ensuring a comprehensive understanding of the baseline security measures in place. Moreover, the system study involves conducting detailed vulnerability assessments and penetration testing exercises.

These activities are pivotal in pinpointing potential security gaps and vulnerabilities within the web application environment. Vulnerability assessments entail scanning web servers, applications, and third-party plugins for known vulnerabilities, while penetration testing simulates real-world attacks to evaluate the resilience of existing security measures. This phase aims to uncover exploitable entry points and weaknesses that could be exploited by malicious actors, providing critical insights into areas requiring immediate attention and mitigation.

Furthermore, the system study delves into analyzing the evolving threat landscape in web application security. This includes staying abreast of current trends in cyber threats, such as advanced persistent threats (APTs) and zero-day vulnerabilities, which pose significant challenges to traditional security measures. Understanding these emerging threats enables a tailored implementation of security measures, ensuring that defenses are adapted to the latest threat landscape and vulnerabilities.

ModSecurity Firewall effectively, ensuring it addresses specific vulnerabilities and mitigates risks proactively. Stakeholder requirements and expectations are also gathered during this phase, ensuring that the objectives align with organizational goals for enhanced security and compliance with regulatory standards.

Feasibility study

The feasibility study phase begins with an assessment of technical feasibility, focusing on evaluating the capability of integrating ModSecurity Firewall with Threat Intelligence within the organization’s existing infrastructure. This includes analyzing compatibility with current network architecture, hardware resources, and software dependencies. The goal is to ensure that the implementation can be seamlessly integrated without disrupting existing operations or requiring substantial hardware upgrades.

Financial feasibility is another critical aspect addressed in this phase. It involves conducting a cost-benefit analysis to determine the financial viability of implementing ModSecurity Firewall with Threat Intelligence. This includes estimating initial setup costs, ongoing maintenance expenses, and potential cost savings from improved security and reduced incident response efforts. The feasibility study aims to justify the investment in terms of its long-term benefits and return on investment (ROI) for the organization.

Operational feasibility is also evaluated to assess the practicality of implementing andmanaging the ModSecurity Firewall solution within the organization. This includes evaluating the availability of skilled personnel to configure, monitor, and maintain the firewall system effectively. Additionally, the study considers potential operational challenges and risks associated with the integration, such as training requirements for staff and organizational readiness to adopt new security protocols.

Economic Feasibility

The economic feasibility of integrating ModSecurity Firewall with Threat Intelligence involves a comprehensive cost-benefit analysis. Initially, there are costs associated with acquiring and implementing ModSecurity Firewall, including software licenses, hardware upgrades if needed, and professional services for deployment. These initial investments aim to establish robust defenses against web-based attacks like SQL injection and cross-site scripting (XSS), enhancing overall cybersecurity posture.

Moreover, ongoing maintenance costs include regular updates to threat intelligence feeds and continuous monitoring of firewall operations. These operational expenses ensure that the firewall remains effective in detecting and mitigating emerging threats. By quantifying these costs against potential savings from reduced security incidents and operational disruptions, organizations can justify the economic viability of integrating ModSecurity Firewall.

Furthermore, the return on investment (ROI) from integrating ModSecurity Firewall with Threat Intelligence extends beyond cost savings. It includes intangible benefits such as improved customer trust and enhanced brand reputation due to strengthened security measures. The investment in advanced threat detection and prevention capabilities aligns withlong-term strategic goals of maintaining regulatory compliance and safeguarding sensitive data, thus contributing to the organization's overall resilience against cyber threats.

Technical Feasibility

The technical feasibility of integrating ModSecurity Firewall with Threat Intelligence begins with a thorough assessment of compatibility. This involves evaluating whether the organization's current hardware infrastructure can support the installation and operation of ModSecurity Firewall without compromising performance. Compatibility checks extend to software systems, ensuring that the firewall integrates seamlessly with existing web servers,

applications, and backend services. Assessing network architecture is crucial to determine if the infrastructure can handle the increased traffic and processing demands introduced by the firewall and threat intelligence feeds.

Scalability is another critical consideration in the technical feasibility assessment. It involves evaluating ModSecurity Firewall's ability to scale as the organization grows and its security needs evolve. Assessments should include plans for horizontal scalability, such as adding more servers or instances, and vertical scalability, such as upgrading hardware resources to accommodate increased workload and traffic. Additionally, planning for load balancing configurations and ensuring high availability are essential to maintain uninterrupted protection against web-based attacks while minimizing downtime and potential disruptions.

Implementation requirements for integrating ModSecurity Firewall include detailed steps for installation, configuration, and customization. This involves setting up threat intelligence feeds and defining rulesets tailored to the organization's specific security policies and regulatory requirements. Adequate training and skill development for IT personnel responsible for managing and maintaining the firewall system are crucial.

Training should cover configuring rules, monitoring alerts, and effectively responding to security incidents to ensure the firewall operates at peak performance and maximizes threat detection capabilities.

Social Feasibility

Social feasibility for integrating ModSecurity Firewall with Threat Intelligence involves evaluating how the integration aligns with the organization's social environment, stakeholder expectations, and acceptance among users and employees. It starts with understanding the perceptions and attitudes towards cybersecurity within the organization and among its stakeholders. Clear communication about the benefits of enhanced web application security, such as reduced risks of data breaches and improved customer trust, is essential in gaining support and buy-in from stakeholders.

Furthermore, social feasibility assesses the readiness of employees and users to adapt to new security protocols and technologies introduced by ModSecurity Firewall. This includes addressing potential concerns about changes in workflow or user experience due to enhanced security measures. Effective training and awareness programs can help mitigate resistance and ensure that employees understand their roles in maintaining a secure web environment.

Chapter 4
Methodology and Implementation

4.1 Methodology

Methodology revolves around the seamless integration of ModSecurity with a robust threat intelligencefeed to enhance web application security. Initially, ModSecurity is deployed on a high-performance server running Parrot OS, chosen for its advanced security features and tools. The web server, powered by Nginx, serves as the primary platform where ModSecurity intercepts and inspectsincoming HTTP requests. To ensure real-time updates and adaptive threat detection, threat intelligencefeed is integrated, enabling ModSecurity to dynamically update its rules and detect emerging threats. Additionally, a comprehensive logging and monitoring framework is established using a Security Information and Event Management (SIEM) system to correlate security events and provide actionableinsights. This setup is rigorously tested in a controlled environment to fine-tune the configurations andensure optimal performance. The final phase involves the deployment in a production environment, followed by continuous monitoring and periodic updates to maintain robust security posture.

4.2 Algorithms

In the dynamic and interconnected world of today, cybersecurity has become a cornerstone of digital resilience, essential for protecting sensitive data, preserving operational continuity, and mitigating risks posed by malicious actors. Central to effective cybersecurity strategies are robust methodologies that guide the design, implementation, and management of defenses against evolving threats.

Cybersecurity methodologies encompass a holistic approach to safeguarding digital assets and infrastructure. They integrate proactive measures for threat detection, incident response, and vulnerability management, ensuring organizations can preemptively identify and mitigate potential risks. These methodologies are grounded in continuous monitoring, threat intelligence integration, and adaptive security controls that adapt to emerging threats and vulnerabilities.

Effective cybersecurity methodologies not only prioritize the protection of critical assets but also emphasize compliance with regulatory standards and best practices. They foster a cultureof security awareness and resilience across organizational levels, promoting collaboration between IT teams, stakeholders, and end-users to uphold a unified defense posture.

As cyber threats evolve in sophistication and scale, cybersecurity methodologies play a pivotal role in enabling organizations to stay ahead of potential risks, respond decisively to incidents, and maintain trust and reliability in their digital operations. By embracing comprehensive cybersecurity methodologies, organizations can fortify their resilience against cyber threats while navigating the complexities of a digitally interconnected landscape.

4.2.1 Signature Based Detection

Signature-Based Detection is a fundamental approach in cybersecurity that operates by comparing incoming network traffic against a database of predefined signatures. These signatures are crafted based on known patterns and behaviors associated with malicious activities such as malware, exploits, and common attack vectors like SQL injection or cross-site scripting (XSS). When a packet matches a signature, it indicates a high likelihood of malicious intent, prompting immediate action by security systems.

In practice, systems like ModSecurity, deployed on robust server environments such as Parrot OS with Nginx, utilize signature-based detection to analyze HTTP requests in real-time. This method enables swift identification and mitigation of known threats by either blocking suspicious traffic or generating alerts for further investigation. Signature-based detection is highly effective in defending against well-established threats, offering a proactive defense mechanism that is essential for protecting web applications and networks from routineexploits.

However, the efficacy of signature-based detection is limited to detecting only those threats for which signatures have been predefined. It may struggle with detecting novel or zero-day attacks that do not match existing signatures. Continuous updates and maintenance of signature databases are crucial to keep pace with evolving cyber threats and ensure effective threat detection capabilities.

4.2.2 Rule-Based Detection

Rule-Based Detection complements signature-based detection by offering a flexible framework to evaluate network traffic based on predefined rules rather than specific signatures. These rules define conditions under which traffic should be allowed, blocked, or flagged for further analysis. Unlike signatures, which focus on specific attack patterns, rules consider various attributes of network packets such as source and destination IP addresses, ports, HTTP methods, and payload content.

Organizations leverage rule-based detection within their Intrusion Detection and Prevention Systems (IDPS) to enforce security policies and respond dynamically to emerging threats. By integrating threat intelligence data, organizations can continuously update and refine rules to counteract evolving attack vectors effectively. This adaptive approach enhances the agility of security systems, enabling rapid responses to new and unknown threats that may evade traditional signature-based detection.

Rule-based detection is instrumental in enhancing the granularity of traffic filtering and mitigating risks associated with sophisticated cyber threats. Its dynamic nature allows organizations to tailor security measures to their specific operational environments and regulatory requirements, ensuring comprehensive protection against a wide range of potential vulnerabilities and attack scenarios.

4.3 Implementation

The implementation of the Dynamic Threat Intelligence Integration (DTII) methodology involves several critical steps to ensure seamless integration of the ModSecurity Web Application Firewall (WAF) with the AlienVault OTX threat intelligence feed. This process is designed to enhance the WAF's ability to detect and mitigate emerging cyber threats by dynamically updating its rule sets.

4.3.1 ModSecurity Installation and Configuration

The first step in the implementation is the installation and configuration of ModSecurity on the web server. ModSecurity, an open-source WAF, is integrated with the web application server, such as Apache or Nginx, to monitor and filter HTTP traffic. This involves downloading the ModSecurity module, compiling it with the web server, and configuring it to start filtering traffic.

Install VirtualBox:

Download and install VirtualBox from the official website.

Follow the installation instructions for the operating system.

Download Parrot OS file:

Go to the Parrot OS official website and download the ISO file.

Create a new virtual machine in VirtualBox:

Open VirtualBox and click on "New."

Name the VM and select the type as "Linux" and version as "Debian (64-bit)."

Allocate memory (RAM) (e.g., 2048 MB).

Create a virtual hard disk and allocate space (e.g., 20 GB).

Follow the prompts to complete the VM creation process.

Install Parrot OS:

Start the VM and select the downloaded Parrot OS ISO file to boot from.

Follow the installation instructions provided by Parrot OS.

After installation, log in with the default credentials and change the password.

Update the package lists and install necessary libraries:

Using the following command:

```
sudo apt-get install bison build-essential ca-certificates curl dh-autoreconf doxygen \
  flex gawk git iputils-ping libcurl4-gnutls-dev libexpat1-dev libgeoip-dev liblmdb-dev \
  libpcre3-dev libpcre++-dev libssl-dev libtool libxml2 libxml2-dev libyajl-dev locales \
  lua5.3-dev pkg-config wget zlib1g-dev zlibc libxslt libgd-dev
```

Installation and Compilation of ModSecurity

Download git

```
sudo apt install git
```

Clone the ModSecurity Github repository from the /opt directory:

```
cd /opt && sudo git clone https://github.com/SpiderLabs/ModSecurity
```

Change the directory to the ModSecurity directory:

```
cd ModSecurity
```

Run the following git commands to initialize and update the submodule:

```
sudo git submodule init
sudo git submodule update
```

Run the build.sh script:

```
sudo ./build.sh
```

Run the configure file for getting all dependencies for the buildprocess:

```
sudo ./configure
```

Run the make command to build ModSecurity:

```
sudo make
```

Install ModSecurity by running the following command:

```
sudo make install
```

Downloading ModSecurity-Nginx Connector:

Clone the Nginx-connector from the /opt directory:

```
cd /opt && sudo git clone --depth 1 https://github.com/SpiderLabs/ModSecurity-nginx.git
```

Building the ModSecurity Module For Nginx:

Download Nginx into the /opt directory:

```
cd /opt && sudo wget http://nginx.org/download/nginx-1.14.0.tar.gz
```

Extract the files:

```
sudo tar -xvzmf nginx-1.14.0.tar.gz
```

Move to extracted directory:

```
cd nginx-1.14.0
```

Check the version of Nginx:

```
nginx -V
```

Compile the ModSecurity Module:

```
sudo ./configure
```

Build the modules:

```
sudo make modules
```

Create a directory for ModSecurity Module in Nginx configuration folder:

```
sudo mkdir /etc/nginx/modules
```

Move to extracted directory:

```
cd nginx-1.14.0
```

Copy the compiled ModSecurity Module into Nginx configuration folder:

```
sudo cp objs/ngx_http_modsecurity_module.so /etc/nginx/modules
```

Loading ModSecurity Module in Nginx

Open the Nginx configuration file and add the following content:

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
load_module /etc/nginx/modules/ngx_http_modsecurity_module.so;
```

Setting Up OWASP-CRS

Clone the OWASP-CRS GitHub repository:

```
sudo git clone https://github.com/coreruleset/coreruleset /usr/local/modsecurity-crs
```

Rename the crs-setup.conf.example to crs-setup.conf:

```
sudo mv /usr/local/modsecurity-crs/crs-setup.conf.example /usr/local/modsecurity-crs/crs-setup.conf
```

Configuring ModSecurity

Create a ModSecurity directory in /etc/nginx/:

```
sudo mkdir -p /etc/nginx/modsec
```

Copy the Unicode mapping file and ModSecurity configurationfile from cloned GitHub repository:

```
sudo cp /opt/ModSecurity/unicode.mapping /etc/nginx/modsec
sudo cp /opt/ModSecurity/modsecurity.conf-recommended /etc/nginx/modsec/modsecurity.conf
```

Remove the .recommended extension from the ModSecurityconfiguration file:

```
sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf
```

Change the value of SecRuleEngine to On in ModSecurityconfiguration file:

```
SecRuleEngine On
```

Create a new configuration file called main.conf:

```
sudo touch /etc/nginx/modsec/main.conf
```

Specify the rules and the ModSecurity configuration file forNgi

```
Include /etc/nginx/modsec/modsecurity.conf
Include /usr/local/modsecurity-crs/crs-setup.conf
Include /usr/local/modsecurity-crs/rules/*.conf
```

Configuring Nginx:

Enable ModSecurity in site configuration file by opening the /etc/nginx/sites-available/default:

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;

        modsecurity on;
        modsecurity_rules_file /etc/nginx/modsec/main.conf;

        index index.html index.htm index.nginx-debian.html;

        server_name _;
        location / {
                try_files $uri $uri/ =404;
        }
}
```

Restart the nginx service to apply the configuration:

```
sudo systemctl restart nginx
```

4.3.2 AlienVault OTX Integration

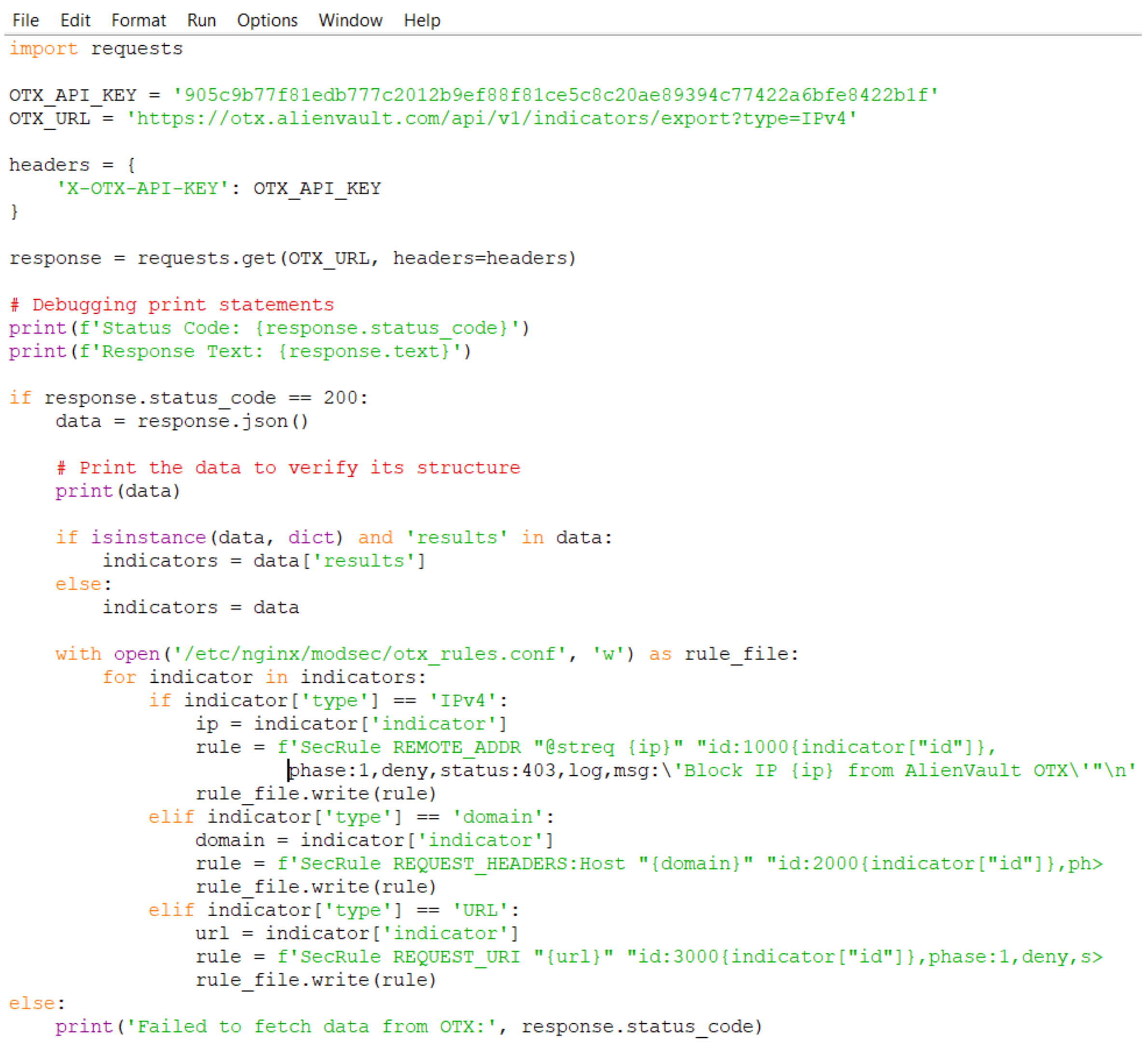
To integrate threat intelligence feeds, an AlienVault OTX account is created. This involves registering on the AlienVault OTX platform and obtaining API keys required for accessing the threat intelligence feeds. These API keys allow the system to pull real-time data on various cyber threats, including indicators of compromise (IOCs), malicious IP addresses, URLs, and attack patterns.

4.3.3 Data Aggregation and Normalization

With the API keys configured, the system starts collecting threat data from AlienVault OTX. The collected data is aggregated from multiple sources within the platform to ensure a comprehensive threat landscape view. This data is then normalized, which involves converting it into a consistent format that can be used to develop ModSecurity rules. Normalization ensures compatibility and effectiveness of the security rules derived from the threat intelligence data.

4.3.4 Custom Rule Development

Based on the normalized threat intelligence data, custom ModSecurity rules are developed. These rules target specific threats identified during the data collection phase. The rules are written in the ModSecurity rule syntax and are designed to detect and block malicious traffic patterns, such as SQL injections, cross-site scripting (XSS), and remote file inclusions.



```
import requests

OTX_API_KEY = '905c9b77f81edb777c2012b9ef88f81ce5c8c20ae89394c77422a6bfe8422b1f'
OTX_URL = 'https://otx.alienvault.com/api/v1/indicators/export?type=IPv4'

headers = {
    'X-OTX-API-KEY': OTX_API_KEY
}

response = requests.get(OTX_URL, headers=headers)

# Debugging print statements
print(f'Status Code: {response.status_code}')
print(f'Response Text: {response.text}')

if response.status_code == 200:
    data = response.json()

    # Print the data to verify its structure
    print(data)

    if isinstance(data, dict) and 'results' in data:
        indicators = data['results']
    else:
        indicators = data

    with open('/etc/nginx/modsec/otx_rules.conf', 'w') as rule_file:
        for indicator in indicators:
            if indicator['type'] == 'IPv4':
                ip = indicator['indicator']
                rule = f'SecRule REMOTE_ADDR "@streq {ip}" "id:1000{indicator["id"]},
                       phase:1,deny,status:403,log,msg:\'Block IP {ip} from AlienVault OTX\'"\n'
                rule_file.write(rule)
            elif indicator['type'] == 'domain':
                domain = indicator['indicator']
                rule = f'SecRule REQUEST_HEADERS:Host "{domain}" "id:2000{indicator["id"]},ph>
                rule_file.write(rule)
            elif indicator['type'] == 'URL':
                url = indicator['indicator']
                rule = f'SecRule REQUEST_URI "{url}" "id:3000{indicator["id"]},phase:1,deny,s>
                rule_file.write(rule)
else:
    print('Failed to fetch data from OTX:', response.status_code)
```

Figure 4.1: Python Script for rule conversion

The Python script in figure 4.1 is designed to interact with the AlienVault Open Threat Exchange (OTX) API to retrieve threat intelligence data and use it to create security rules for ModSecurity, a popular web application firewall. The script begins by importing the requests library, which is used for making HTTP requests. The OTX API key is stored in the OTX_API_KEY variable, and the OTX API endpoint URL for exporting IPv4 indicators is stored in the OTX_URL variable.

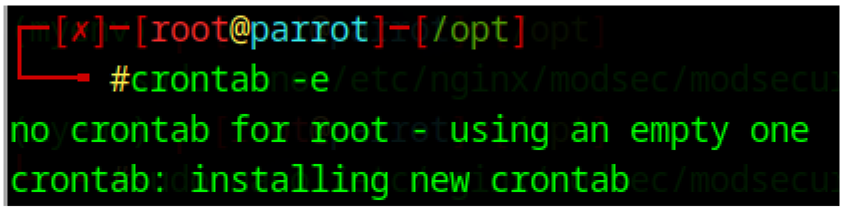
A dictionary named headers is created to include the necessary API key in the request's header. The script then sends a GET request to the OTX API using these headers. The response is captured in the response variable. If the response status code is 200 (indicating success), the script converts the response data from JSON format into a Python dictionary or list.

To ensure the data structure is correct, the script checks if the response is a dictionary containing a 'results' key. It then iterates through the threat indicators, generating ModSecurity rules based on the type of indicator—IPv4, domain, or URL. These rules are written to a configuration file (/etc/nginx/modsec/otx_rules.conf) for ModSecurity to enforce. If the API request fails, the script outputs an error message with the failed status code.

4.3.5 Dynamic Rule Update Mechanism

To ensure that ModSecurity can adapt to new threats in real-time, a dynamic rule update mechanism is implemented. This involves developing scripts or using existing plugins that periodically fetch new threat intelligence data from AlienVault OTX. The fetched data is then automatically converted into ModSecurity rule format and applied to the WAF configuration. This automation allows ModSecurity to continuously update its rule sets without requiring manual intervention, ensuring up-to-date protection against the latest threats.

To automate the process of updating security rules, you can create a cron job as in figure 4.2 that runs a specific script periodically. This ensures that your rules are always up-to-date without requiring manual intervention. Start by opening the crontab configuration file for editing by executing the command crontab -e in your terminal.



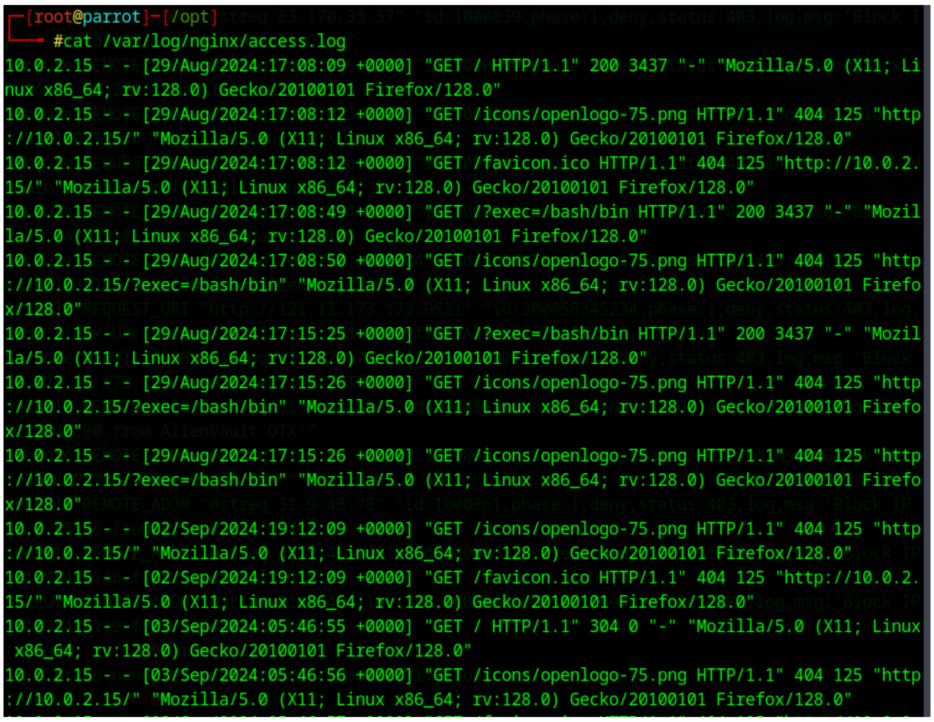
```
┌─[✗]─[root@parrot]─[/opt]
└──╼ #crontab -e
no crontab for root - using an empty one
crontab: installing new crontab
```

Figure 4.2 : Scheduling updates using crontab

Once inside the crontab editor, add a new line to schedule the script to run at a specified interval. For instance, to run the script daily at midnight, you would add the following line: 0 0 * * * /usr/bin/python3 /path/to/your/otx_script.py. This configuration schedules the script to execute automatically at 00:00 hours every day, effectively maintaining the latest rules in your system.

4.3.6 Logging and Monitoring

Logging and monitoring are crucial aspects of the implementation to ensure the operational health and security effectiveness of the integrated system. Logging mechanisms are configured to record all ModSecurity activities, including detected threats, blocked requests, and rule updates. This logging provides a detailed audit trail for forensic analysis and compliance purposes.



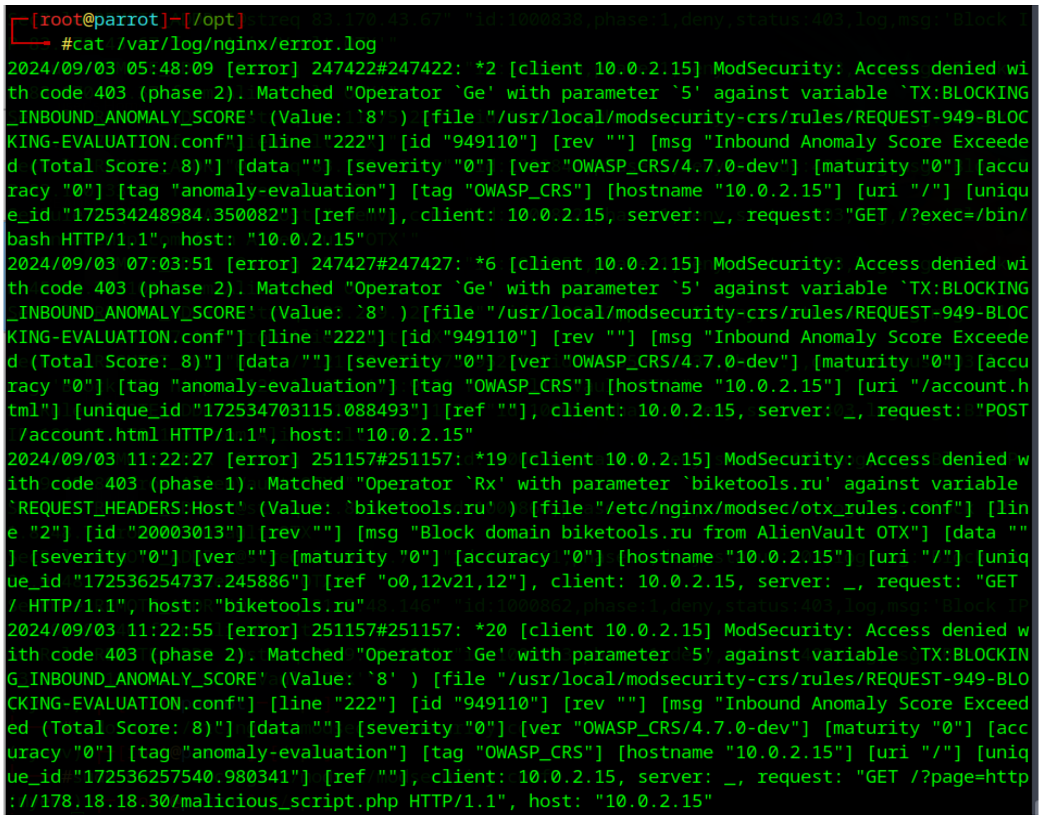
```
┌─[root@parrot]─[/opt]
└──╼ #cat /var/log/nginx/access.log
10.0.2.15 - - [29/Aug/2024:17:08:09 +0000] "GET / HTTP/1.1" 200 3437 "-" "Mozilla/5.0 (X11; Li
nux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [29/Aug/2024:17:08:12 +0000] "GET /icons/openlogo-75.png HTTP/1.1" 404 125 "http
://10.0.2.15/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [29/Aug/2024:17:08:12 +0000] "GET /favicon.ico HTTP/1.1" 404 125 "http://10.0.2.
15/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [29/Aug/2024:17:08:49 +0000] "GET /?exec=/bash/bin HTTP/1.1" 200 3437 "-" "Mozil
la/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [29/Aug/2024:17:08:50 +0000] "GET /icons/openlogo-75.png HTTP/1.1" 404 125 "http
://10.0.2.15/?exec=/bash/bin" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefo
x/128.0"
10.0.2.15 - - [29/Aug/2024:17:15:25 +0000] "GET /?exec=/bash/bin HTTP/1.1" 200 3437 "-" "Mozil
la/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [29/Aug/2024:17:15:26 +0000] "GET /icons/openlogo-75.png HTTP/1.1" 404 125 "http
://10.0.2.15/?exec=/bash/bin" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefo
x/128.0"
10.0.2.15 - - [29/Aug/2024:17:15:26 +0000] "GET /icons/openlogo-75.png HTTP/1.1" 404 125 "http
://10.0.2.15/?exec=/bash/bin" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefo
x/128.0"
10.0.2.15 - - [02/Sep/2024:19:12:09 +0000] "GET /icons/openlogo-75.png HTTP/1.1" 404 125 "http
://10.0.2.15/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [02/Sep/2024:19:12:09 +0000] "GET /favicon.ico HTTP/1.1" 404 125 "http://10.0.2.
15/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [03/Sep/2024:05:46:55 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux
 x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
10.0.2.15 - - [03/Sep/2024:05:46:56 +0000] "GET /icons/openlogo-75.png HTTP/1.1" 404 125 "http
://10.0.2.15/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
```

Fig 4.3 Access Logs

The provided access log entries in the figure 4.3 from Nginx detail a series of HTTP requests made by a client with the IP address 10.0.2.15 using the Firefox browser on a Linux system. The logs show requests to various URLs, including the root ("/") and some specific paths like /?exec=/bash/bin and /?exec=/bin/bash. Most of these requests resulted in a successful 200 status code, meaning the server processed them without errors. However, certain requests attempted to access non-existent resources, such as openlogo-75.png and favicon.ico, which led to 404 errors indicating "Not Found". Additionally, the log records a request to /?exec=/bin/bash that was denied with a 403-status code, likely due to security measures implemented by the server.

The provided log entries in the figure 4.4 from the Nginx error log indicate that ModSecurity, a web application firewall, is actively blocking requests to the server due to detected security violations. Each entry details an incident where a client's request was denied with a 403-status code, indicating forbidden access. The first and second entries show that ModSecurity blocked requests because the "Inbound Anomaly Score" exceeded a threshold, as determined by a set of rules from the OWASP

Core Rule Set (CRS). The anomaly score, in these cases, reached 8, triggering the blocking mechanism.



```
┌─[root@parrot]─[/opt]
└──╼ #cat /var/log/nginx/error.log
2024/09/03 05:48:09 [error] 247422#247422: *2 [client 10.0.2.15] ModSecurity: Access denied wi
th code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING
_INBOUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOC
KING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceede
d (Total Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.7.0-dev"] [maturity "0"] [accu
racy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/"] [uniqu
e_id "172534248984.350082"] [ref ""], client: 10.0.2.15, server: _, request: "GET /?exec=/bin/
bash HTTP/1.1", host: "10.0.2.15"
2024/09/03 07:03:51 [error] 247427#247427: *6 [client 10.0.2.15] ModSecurity: Access denied wi
th code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING
_INBOUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOC
KING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceede
d (Total Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.7.0-dev"] [maturity "0"] [accu
racy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/account.h
tml"] [unique_id "172534703115.088493"] [ref ""], client: 10.0.2.15, server: _, request: "POST
 /account.html HTTP/1.1", host: "10.0.2.15"
2024/09/03 11:22:27 [error] 251157#251157: *19 [client 10.0.2.15] ModSecurity: Access denied w
ith code 403 (phase 1). Matched "Operator `Rx' with parameter `biketools.ru' against variable 
`REQUEST_HEADERS:Host' (Value: `biketools.ru' ) [file "/etc/nginx/modsec/otx_rules.conf"] [lin
e "2"] [id "20003013"] [rev ""] [msg "Block domain biketools.ru from AlienVault OTX"] [data ""
] [severity "0"] [ver ""] [maturity "0"] [accuracy "0"] [hostname "10.0.2.15"] [uri "/"] [uniq
ue_id "172536254737.245886"] [ref "o0,12v21,12"], client: 10.0.2.15, server: _, request: "GET 
/ HTTP/1.1", host: "biketools.ru"
2024/09/03 11:22:55 [error] 251157#251157: *20 [client 10.0.2.15] ModSecurity: Access denied w
ith code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKIN
G_INBOUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLO
CKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceed
ed (Total Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.7.0-dev"] [maturity "0"] [acc
uracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/"] [uniq
ue_id "172536257540.980341"] [ref ""], client: 10.0.2.15, server: _, request: "GET /?page=http
://178.18.18.30/malicious_script.php HTTP/1.1", host: "10.0.2.15"
```

Fig 4.4 Error Logs

4.3.7 System Integration and Traffic Filtering

The dynamic rule update mechanism is integrated with the ModSecurity configuration. This setup enables ModSecurity to analyze incoming HTTP traffic in real-time, applying the latest rules to detect and filter out malicious requests. The integration is tested in a controlled environment to ensure that it functions correctly and efficiently under real-world conditions.

Referer Header Manipulation Attack:

The command curl -X GET "http://10.0.2.15" --referer "http://blizko.org" is an example of a referer header manipulation attack. This type of attack exploits the Referer HTTP header, which is used by web browsers to indicate the address (URL) of the webpage that linked to the resource being requested. When a user clicks a link on a webpage, the browser sends the address of the current page as the referer in the HTTP request to the new page.

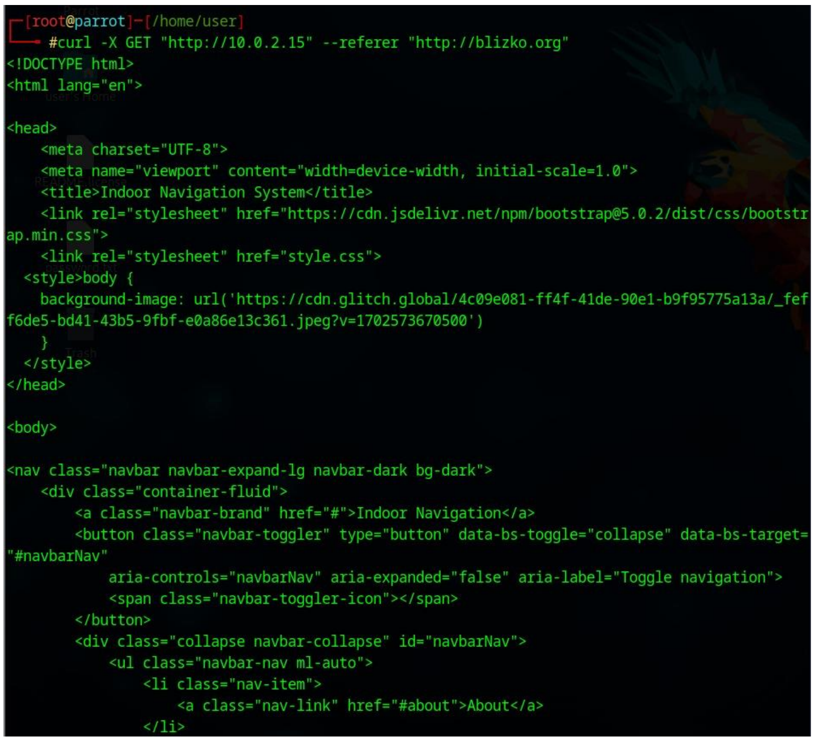
In a referer header manipulation attack, an attacker intentionally alters the referer header to mislead the server into believing the request originated from a trusted or different source. This can be done to bypass security controls, perform unauthorized actions, or simply gather information about how the server handles different referers.

Impacts of the Attack:

1. Bypassing Security Controls: Many web applications rely on the referer header for enforcing security policies, such as ensuring that sensitive actions (like form submissions) only occur from trusted pages. By manipulating the referer, an attacker can trick the server into processing requests that it would otherwise block.
2. Exploiting Trust Relationships: Some web applications grant additional privileges or trust to requests coming from specific referers, especially those within the same domain. An attacker can forge the referer to escalate privileges or gain unauthorized access.
3. Phishing and Social Engineering: By spoofing the referer, an attacker can make it appear as though a request is coming from a legitimate and trusted site, potentially tricking users or the server into divulging sensitive information.
4. Information Gathering: An attacker can manipulate the referer header to see how the server responds to requests from different sources, potentially revealing information about the server's trust relationships, referral tracking, or access controls.

In summary, referer header manipulation is a subtle but powerful attack vector that can lead to security breaches if a web application relies heavily on referer-based logic for authentication, authorization, or other security-related decisions. Proper validation and handling of HTTP headers are essential to mitigate such attacks.

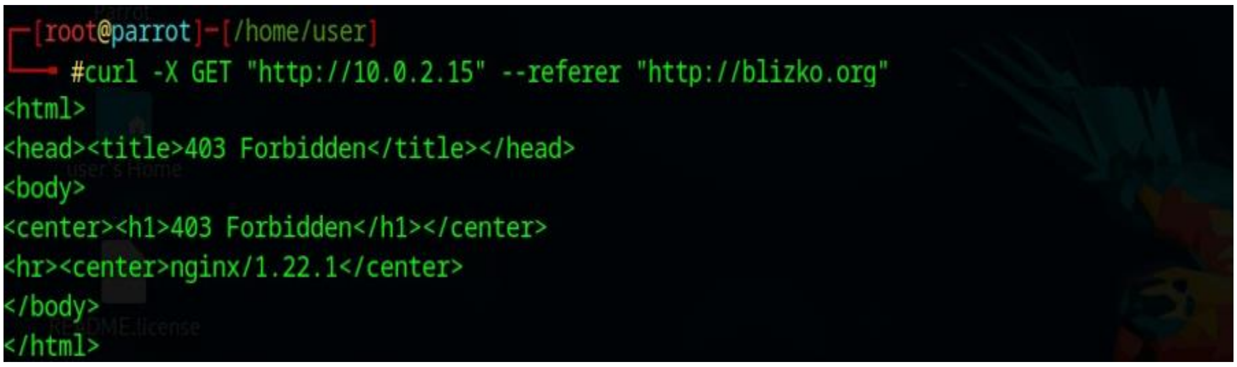
Simulating the attack:



```
┌─[root@parrot]─[/home/user]
└──╼ #curl -X GET "http://10.0.2.15" --referer "http://blizko.org"
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Indoor Navigation System</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstr
ap.min.css">
    <link rel="stylesheet" href="style.css">
  <style>body {
    background-image: url('https://cdn.glitch.global/4c09e081-ff4f-41de-90e1-b9f95775a13a/_fef
f6de5-bd41-43b5-9fbf-e0a86e13c361.jpeg?v=1702573670500')
    }
  </style>
</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Indoor Navigation</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=
"#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item">
                    <a class="nav-link" href="#about">About</a>
                </li>
```

Figure 4.5: Launching the attack before integration

The image (Figure 4.5) shows a terminal session on a Parrot OS system where a user has executed a curl command to send an HTTP GET request to the IP address http://10.0.2.15 while spoofing the Referer header with the value "http://blizko.org". The command used is visible at the top of the terminal: curl -X GET "http://10.0.2.15" --referer "http://blizko.org". The terminal then displays the HTML response returned by the server, indicating that the request successfully retrieved the HTML content of the web page hosted on 10.0.2.15. The HTML content includes a document with a DOCTYPE declaration, an HTML structure with head and body sections.



```
┌─[root@parrot]─[/home/user]
└──╼ #curl -X GET "http://10.0.2.15" --referer "http://blizko.org"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.22.1</center>
</body>
</html>
```

Figure 4.6: Launching the attack after integration

The figure 4.6 shows a terminal session on a Parrot OS system where a user has executed a curl command post integration. The terminal then displayed HTML response returned by the server, indicating that the firewall had detected and blocked the IP from accessing the web page.

Chapter 5
Testing and Validation

Testing and validation are essential processes in software and security systems to ensure that they function as intended and meet predefined requirements. Testing involves systematically checking the software or system to identify any defects, vulnerabilities, or performance issues. This can include baseline testing and integrated testing, depending on the development stage. Validation, on the other hand, is the process of confirming that the final product meets the needs and expectations of the end users and complies with any regulatory or business requirements. Together, testing and validation help ensure the reliability, security, and effectiveness of a system before it is deployed in a production environment.

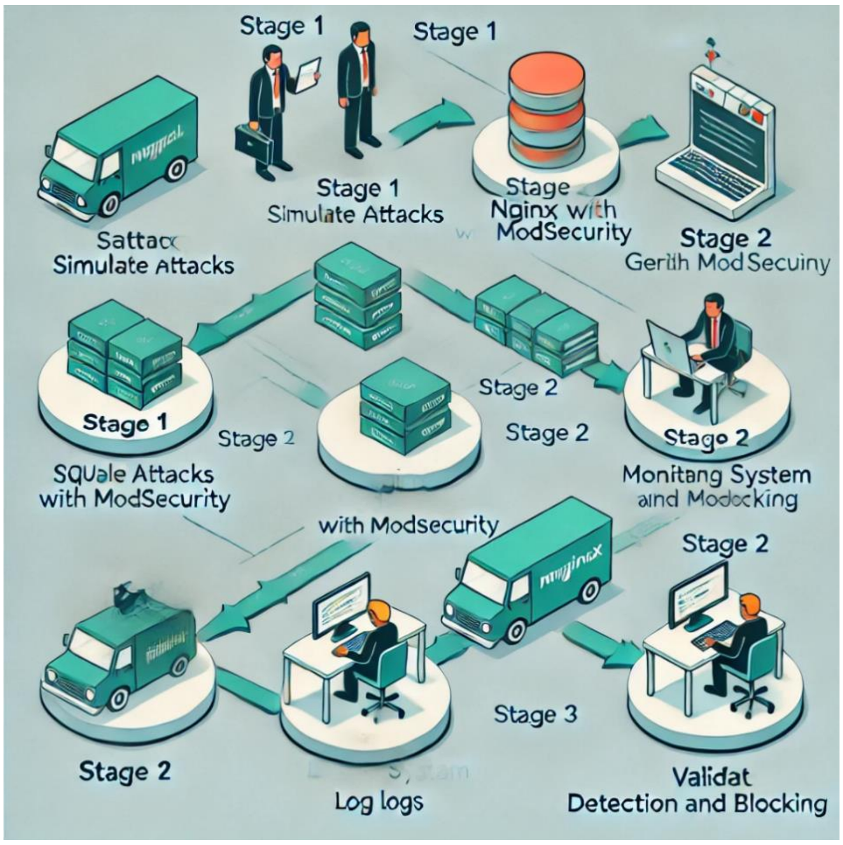


Figure 5.1: Process of Testing and Validation

The UML sequence diagram in figure 5.1 illustrates the process of testing and validation within a system involving multiple entities: an attacker, Nginx with ModSecurity, a logging system, and an administrator. The diagram is divided into three main stages: "Stage 1: Simulate Attacks," "Stage 2: Monitor Logs," and "Stage 3: Validate Detection and Blocking." Each stage is clearly labeled, showing the flow of actions between the entities involved, such as simulating attacks, generating and monitoring logs, and validating the detection and blocking of these attacks.

The diagram uses distinct visual elements to represent each participant's role, with clear arrows indicating the direction of interaction. Despite some typographical errors, the diagram effectively communicates the sequence of activities necessary to ensure that attacks are simulated, logs are monitored, and the system's defenses are validated. The overall design is easy to follow, with appropriate spacing and alignment to enhance readability.

5.1 Baseline Testing

Baseline testing is a crucial process that involves evaluating the performance, security, and functionality of a firewall and web application before they are integrated. This initial phase ensures that both components operate as expected under normal conditions, without the influence of additional security layers. During baseline testing, the firewall is assessed independently to verify its default rules, policies, and response to typical traffic patterns.
Similarly, the web application is tested to identify any inherent vulnerabilities, performance bottlenecks, or issues that could affect its behavior. This process establishes a reference point, allowing for a clear comparison after the integration of additional security measures, and helps in identifying any potential issues that may arise from the integration itself. By conducting thorough baseline testing, the integrity and reliability of the firewall and web application can be ensured, providing a solid foundation for further security enhancements.

Test Cases

Test Case 1: SQL Injection

Objective: Test for SQL injection vulnerability.

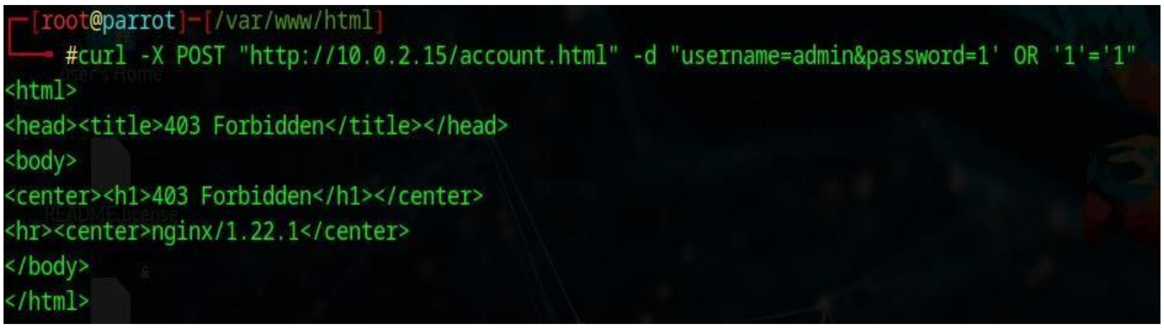
Payload: 1' OR '1'='1

Steps:

Send a login request with an SQL injection payload.

Curl Command:

curl -X POST "http://localhost/account.html" -d "username=admin&password=1' OR '1'='1"



```
┌─[root@parrot]─[/var/www/html]
└──╼ #curl -X POST "http://10.0.2.15/account.html" -d "username=admin&password=1' OR '1'='1"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.22.1</center>
</body>
</html>
```

Test Case 2 : Subdomain Host Header Injection

Objective: Test for Host Header Injection vulnerability by manipulating the Host header in the request.

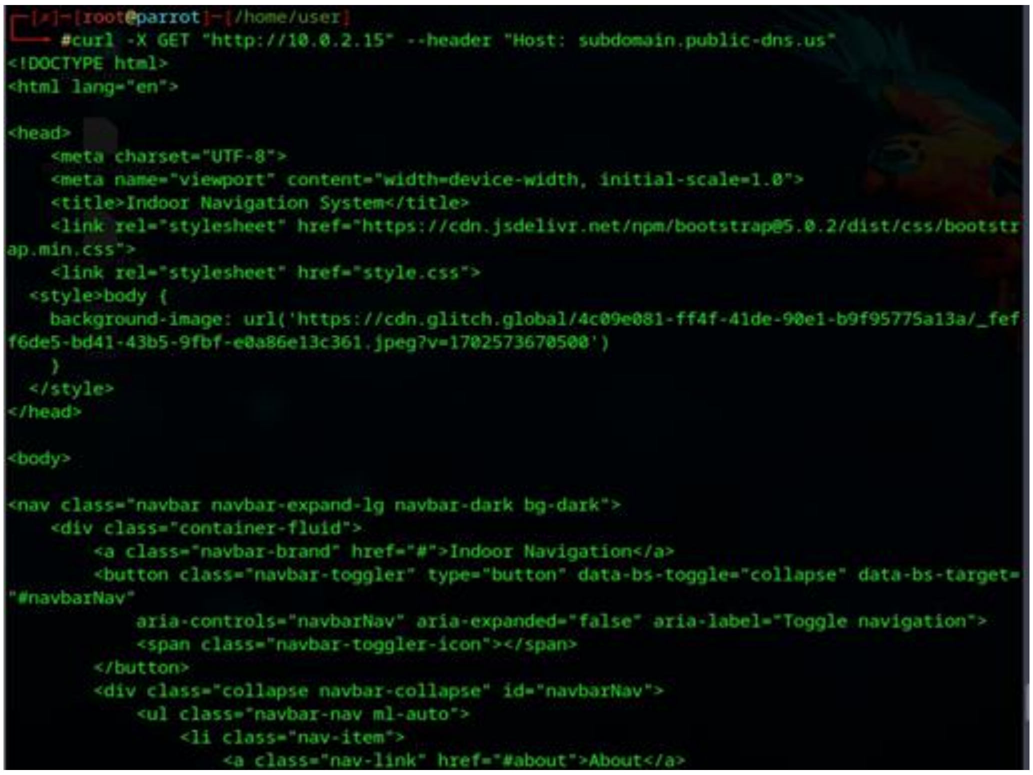
Payload: subdomain.public-dns.us

Steps:

1. Send an HTTP GET request to the target server's IP address.
2. Modify the Host header to a subdomain value that could be used to bypass security controls or redirect to a malicious site.

Curl Command:

curl -X GET "http://10.0.2.15" --header "Host: subdomain.public-dns.us"



```
┌─[✗]─[root@parrot]─[/home/user]
└──╼ #curl -X GET "http://10.0.2.15" --header "Host: subdomain.public-dns.us"
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Indoor Navigation System</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstr
ap.min.css">
    <link rel="stylesheet" href="style.css">
  <style>body {
    background-image: url('https://cdn.glitch.global/4c09e081-ff4f-41de-90e1-b9f95775a13a/_fef
f6de5-bd41-43b5-9fbf-e0a86e13c361.jpeg?v=1702573670500')
    }
  </style>
</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Indoor Navigation</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=
"#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item">
                    <a class="nav-link" href="#about">About</a>
```

Test Case 3: User-Agent Manipulation

Objective: Test for User-Agent header manipulation to identify potential vulnerabilities in the handling of specific user agents by the server.

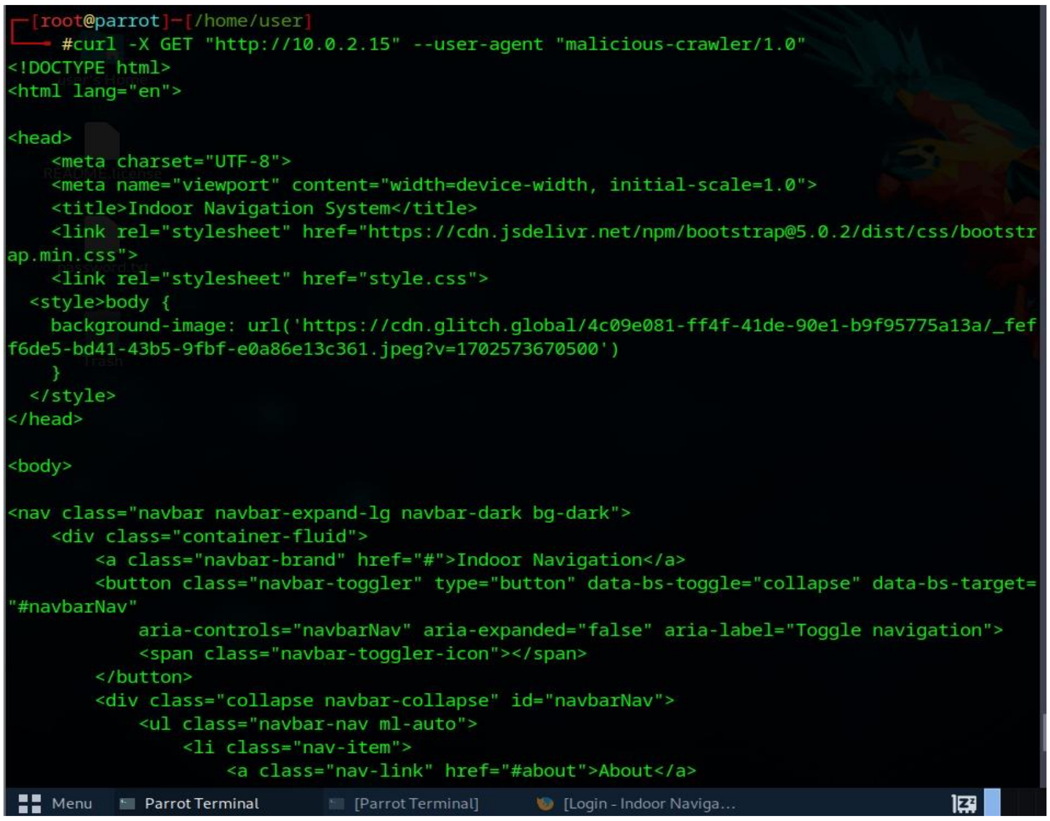
Payload: malicious-crawler/1.0

Steps:

1. Send an HTTP GET request with a manipulated User-Agent header.
2. Observe the server's response to check if it processes or blocks requests from this specific user agent.

Curl Command:

curl -X GET "http://10.0.2.15" --user-agent "malicious-crawler/1.0"



```
┌─[root@parrot]─[/home/user]
└──╼ #curl -X GET "http://10.0.2.15" --user-agent "malicious-crawler/1.0"
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Indoor Navigation System</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstr
ap.min.css">
    <link rel="stylesheet" href="style.css">
  <style>body {
    background-image: url('https://cdn.glitch.global/4c09e081-ff4f-41de-90e1-b9f95775a13a/_fef
f6de5-bd41-43b5-9fbf-e0a86e13c361.jpeg?v=1702573670500')
    }
  </style>
</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Indoor Navigation</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=
"#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item">
                    <a class="nav-link" href="#about">About</a>
```

5.2 Integrated Testing

Integration testing is a critical process that focuses on evaluating the combined performance, security, and functionality of a firewall and web application after they have been integrated. This phase is designed to ensure that the integration does not introduce new vulnerabilities, degrade performance, or cause conflicts between the firewall and the web application. During integration testing, the firewall's ability to protect the web application is rigorously tested by simulating various attack scenarios, checking how well it filters out malicious traffic, and ensuring that legitimate traffic is not mistakenly blocked. Additionally, the web application's functionality is tested to confirm that it continues to operate smoothly with the firewall in place, without experiencing slowdowns or errors. This testing phase helps to identify any issues that may have been introduced during the integration, such as misconfigurations, rule conflicts, or performance degradation. By conducting thorough integration testing, organizations can be confident that their web application is securely protected by the firewall while maintaining optimal performance and usability.

Test Cases:

Test Case 1: SQL Injection

Objective: Test for SQL injection vulnerability.

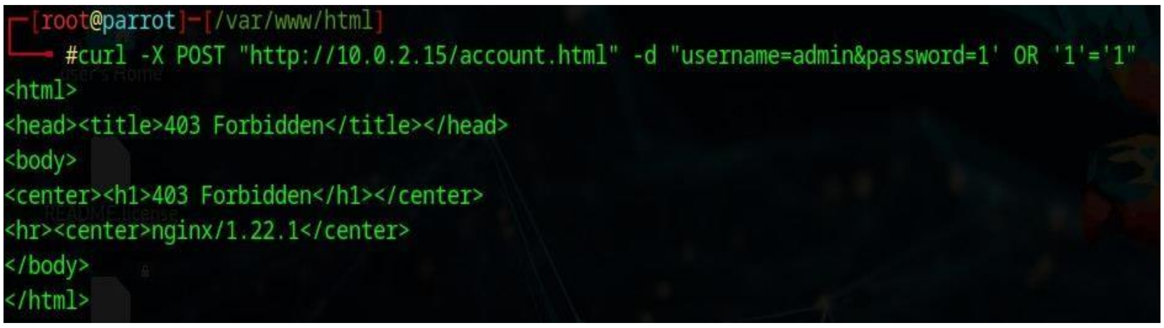
Payload: 1' OR '1'='1

Steps:

Send a login request with an SQL injection payload.

Curl Command:

curl -X POST "http://localhost/account.html" -d "username=admin&password=1' OR '1'='1"



```
┌[root@parrot]─[/var/www/html]
└──╼ #curl -X POST "http://10.0.2.15/account.html" -d "username=admin&password=1' OR '1'='1"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.22.1</center>
</body>
</html>
```

Test Case 2 : Subdomain Host Header Injection

Objective: Test for Host Header Injection vulnerability by manipulating the Host header in the request.

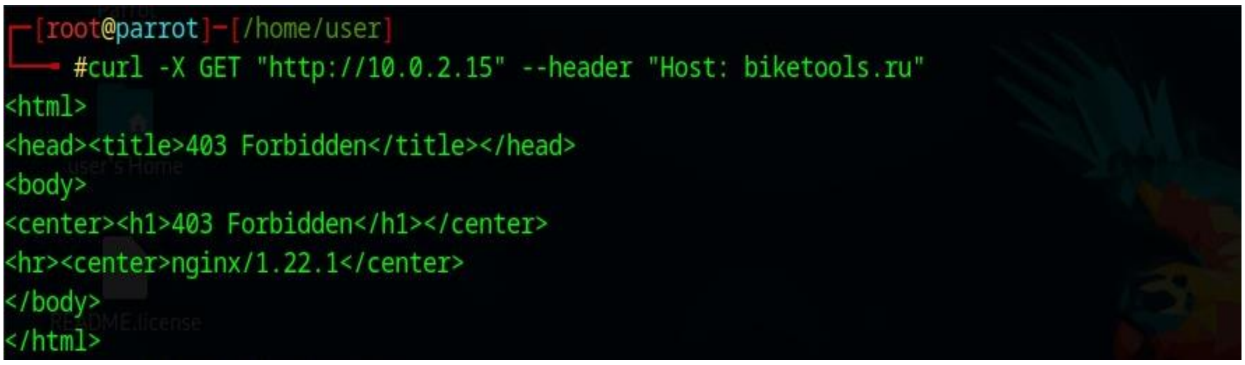
Payload: subdomain.public-dns.us

Steps:

3. Send an HTTP GET request to the target server's IP address.
4. Modify the Host header to a subdomain value that could be used to bypass security controls or redirect to a malicious site.

Curl Command:

curl -X GET "http://10.0.2.15" --header "Host: subdomain.public-dns.us"



```
┌[root@parrot]─[/home/user]
└──╼ #curl -X GET "http://10.0.2.15" --header "Host: biketools.ru"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.22.1</center>
</body>
</html>
```

Test Case 3: User-Agent Manipulation

Objective: Test for User-Agent header manipulation to identify potential vulnerabilities in the handling of specific user agents by the server.

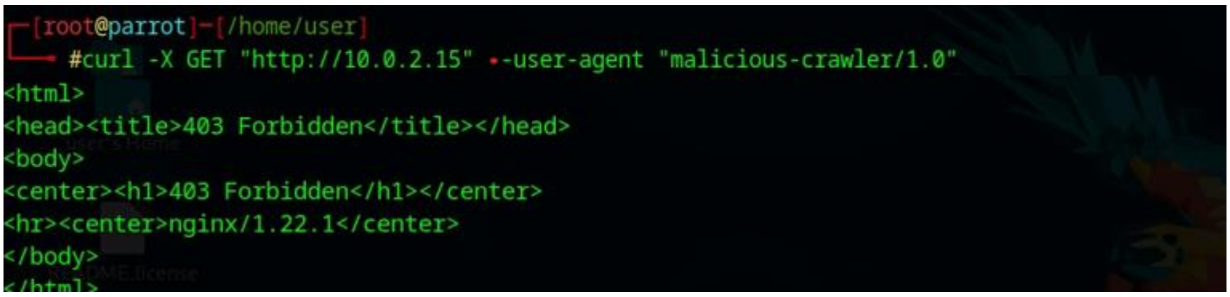
Payload: malicious-crawler/1.0

Steps:

3. Send an HTTP GET request with a manipulated User-Agent header.
4. Observe the server's response to check if it processes or blocks requests from this specific user agent.

Curl Command:

curl -X GET "http://10.0.2.15" --user-agent "malicious-crawler/1.0"

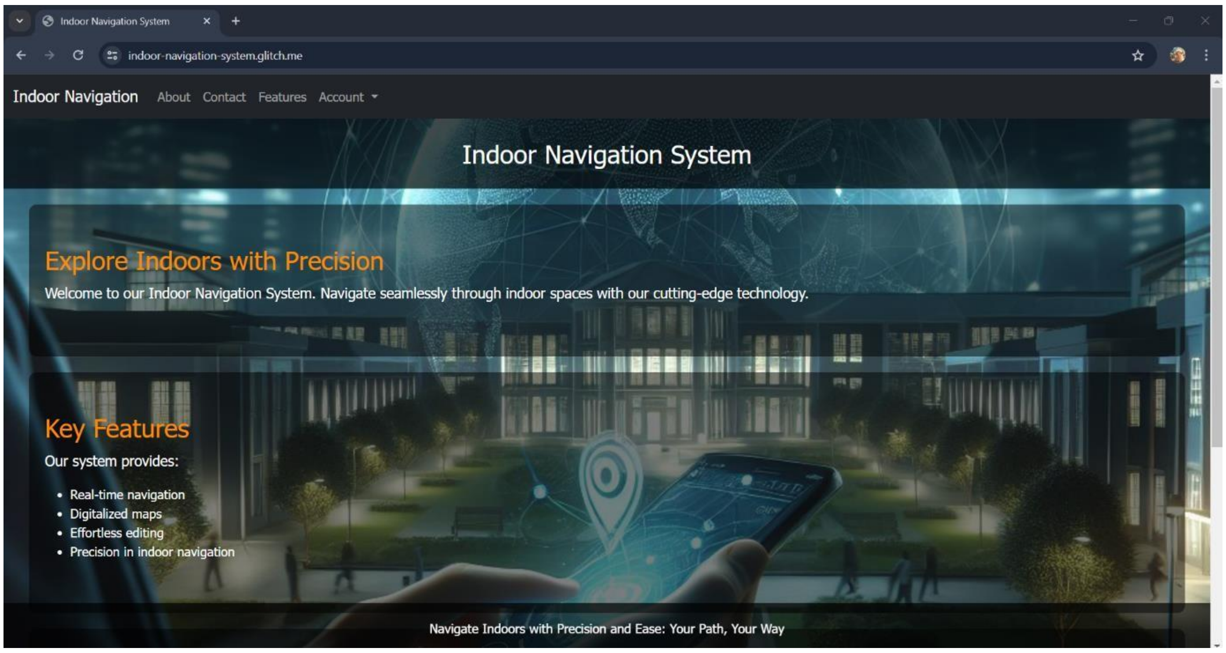


```
┌─[root@parrot]─[/home/user]
└──╼ #curl -X GET "http://10.0.2.15" --user-agent "malicious-crawler/1.0"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.22.1</center>
</body>
</html>
```

5.3 Validation:

In the validation process of firewall integration, the effectiveness of the newly integrated firewall was tested against three different attack scenarios: SQL Injection, Subdomain Host Header Injection, and User-Agent Manipulation. Before integration, only the SQL Injection test case was successfully blocked by the existing firewall, leaving the system vulnerable to the other two attacks. However, after integration, the newly enhanced firewall demonstrated improved security by successfully blocking all three test cases. This result validates the integration process, confirming that the additional security measures are effectively mitigating previously unaddressed vulnerabilities.

Chapter 6
Results



Indoor Navigation System

Explore Indoors with Precision

Welcome to our Indoor Navigation System. Navigate seamlessly through indoor spaces with our cutting-edge technology.

Key Features

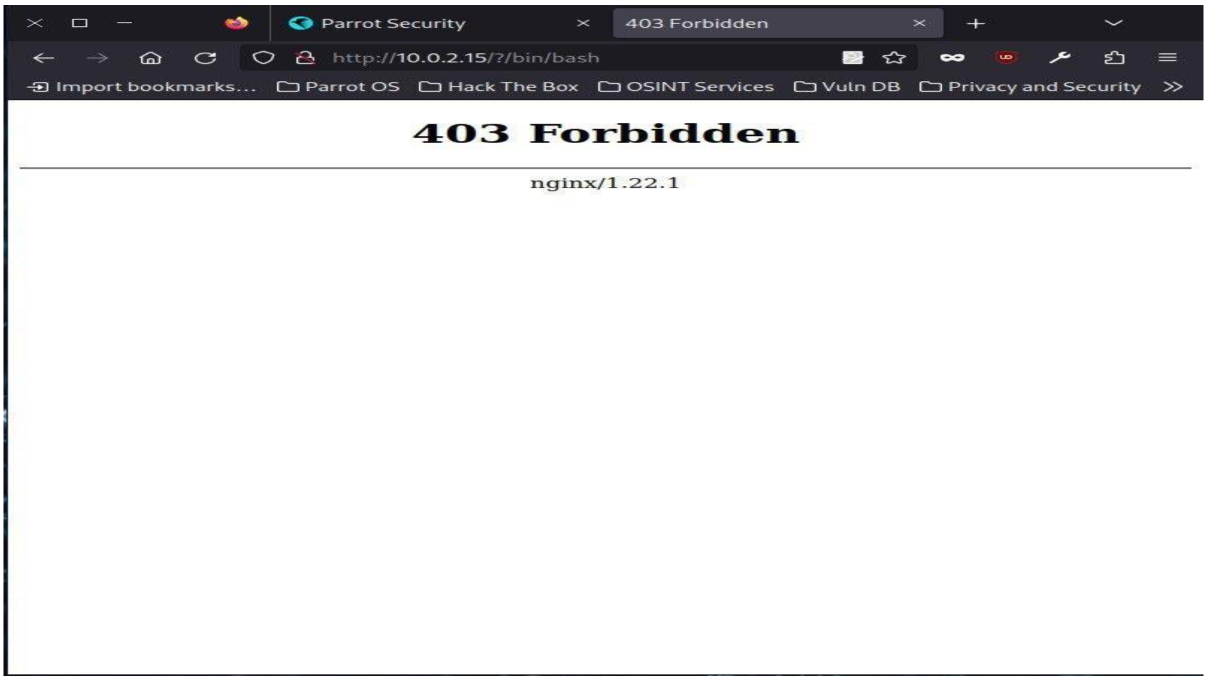
Our system provides:

- Real-time navigation
- Digitalized maps
- Effortless editing
- Precision in indoor navigation

Navigate Indoors with Precision and Ease: Your Path, Your Way

Figure 6.1: Website Before Performing an Attack

The above picture (i.e. figure 6.1) depicts that the website can be accessed before performing theattack to check the firewall setup is successful or not**.**



403 Forbidden

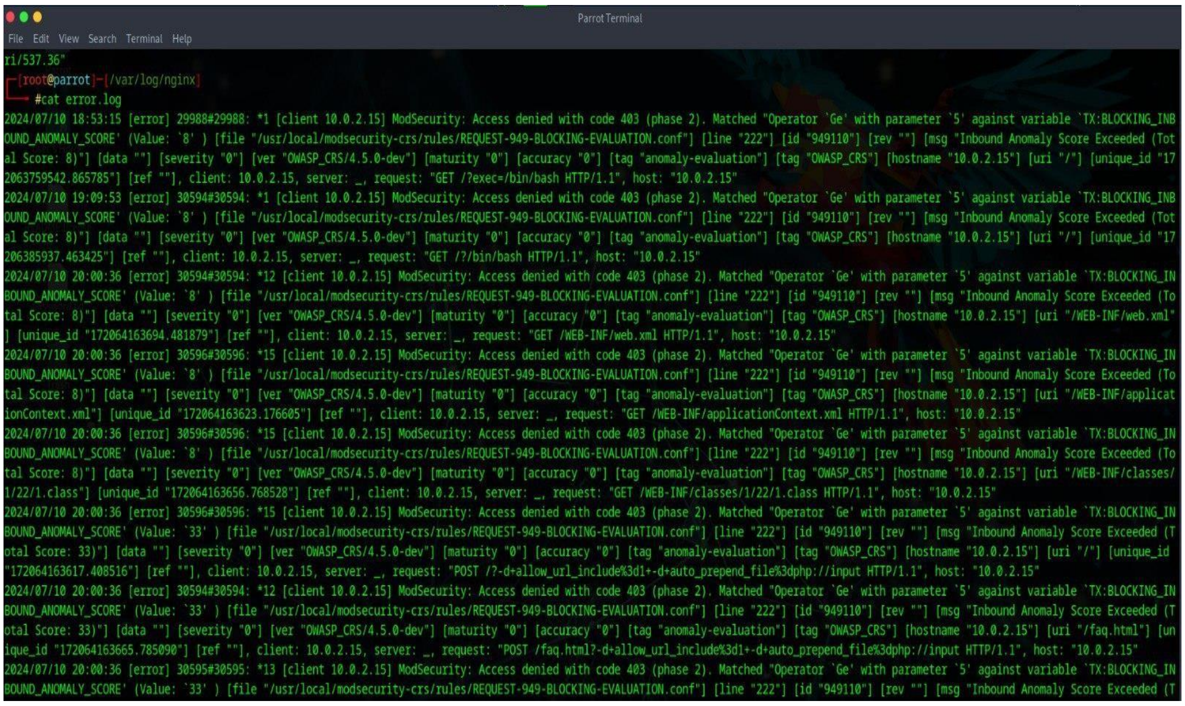
nginx/1.22.1

Figure 6.2: Website After Performing the Attack

The figure 6.2 displays a web browser screen running on Parrot OS, with an HTTP response indicating a "403 Forbidden" error. The URL in the address bar is http://10.0.2.15/?/bin/bash, suggesting an attempt to access a potentially sensitive or restricted path on the web server. The error message clearly states that the request is forbidden, and it is served by an Nginx web server, version 1.22.1.

A "403 Forbidden" status code is returned by the server when the client's request is understoodbut the server refuses to authorize it. This typically occurs due to insufficient permissions to access the requested resource. It could be due to various reasons such as lack of proper authentication credentials, trying to access a directory that is not accessible, or server configurations explicitly denying access to certain files or directories.

The presence of the /bin/bash in the URL indicates an attempt to execute or access the bash shellvia the web server, which is a common tactic in web application attacks. However, the server's configuration effectively prevents this action, enhancing the security posture of the server by blocking unauthorized access to critical system paths.
This error message and the context of the URL suggest a scenario where security measures are appropriately in place to prevent unauthorized access, which is a crucial aspect of maintaining a secure web environment.



```
ri/537.36"
┌─[root@parrot]─[/var/log/nginx]
└──╼ #cat error.log
2024/07/10 18:53:15 [error] 29988#29988: *1 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_INB
OUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (Tot
al Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/"] [unique_id "17
2063759542.865785"] [ref ""], client: 10.0.2.15, server: _, request: "GET /?exec=/bin/bash HTTP/1.1", host: "10.0.2.15"
2024/07/10 19:09:53 [error] 30594#30594: *1 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_INB
OUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (Tot
al Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/"] [unique_id "17
206385937.463425"] [ref ""], client: 10.0.2.15, server: _, request: "GET /?/bin/bash HTTP/1.1", host: "10.0.2.15"
2024/07/10 20:00:36 [error] 30594#30594: *12 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_IN
BOUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (To
tal Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/WEB-INF/web.xml"
] [unique_id "172064163694.481879"] [ref ""], client: 10.0.2.15, server: _, request: "GET /WEB-INF/web.xml HTTP/1.1", host: "10.0.2.15"
2024/07/10 20:00:36 [error] 30596#30596: *15 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_IN
BOUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (To
tal Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/WEB-INF/applicat
ionContext.xml"] [unique_id "172064163623.176605"] [ref ""], client: 10.0.2.15, server: _, request: "GET /WEB-INF/applicationContext.xml HTTP/1.1", host: "10.0.2.15"
2024/07/10 20:00:36 [error] 30596#30596: *15 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_IN
BOUND_ANOMALY_SCORE' (Value: `8' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (To
tal Score: 8)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/WEB-INF/classes/
1/22/1.class"] [unique_id "172064163656.768528"] [ref ""], client: 10.0.2.15, server: _, request: "GET /WEB-INF/classes/1/22/1.class HTTP/1.1", host: "10.0.2.15"
2024/07/10 20:00:36 [error] 30596#30596: *15 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_IN
BOUND_ANOMALY_SCORE' (Value: `33' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (T
otal Score: 33)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/"] [unique_id
"172064163617.408516"] [ref ""], client: 10.0.2.15, server: _, request: "POST /?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input HTTP/1.1", host: "10.0.2.15"
2024/07/10 20:00:36 [error] 30594#30594: *12 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_IN
BOUND_ANOMALY_SCORE' (Value: `33' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (T
otal Score: 33)"] [data ""] [severity "0"] [ver "OWASP_CRS/4.5.0-dev"] [maturity "0"] [accuracy "0"] [tag "anomaly-evaluation"] [tag "OWASP_CRS"] [hostname "10.0.2.15"] [uri "/faq.html"] [un
ique_id "172064163665.785090"] [ref ""], client: 10.0.2.15, server: _, request: "POST /faq.html?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input HTTP/1.1", host: "10.0.2.15"
2024/07/10 20:00:36 [error] 30595#30595: *13 [client 10.0.2.15] ModSecurity: Access denied with code 403 (phase 2). Matched "Operator `Ge' with parameter `5' against variable `TX:BLOCKING_IN
BOUND_ANOMALY_SCORE' (Value: `33' ) [file "/usr/local/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "222"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceeded (T
```

Figure 6.3: Monitoring Errors

The figure 6.3 presents the output from a Parrot Terminal window, displaying the contents of the Nginx error log file located at /var/log/nginx/error.log. The log entries are generated by ModSecurity, a web application firewall (WAF), indicating that several requests were denied due to security rules violations.

Each log entry starts with a timestamp followed by an error code and details about the client making the request. For instance, on 2024/07/10 18:53:15, an error was recorded for a request from the client IP address 10.0.2.15. The ModSecurity module intercepted and denied the request with a 403-status code, which corresponds to "Forbidden." The denial reason is specifiedas "Inbound Anomaly Score Exceeded," with a total score of 8. This indicates that the request triggered multiple security rules, leading to a cumulative anomaly score that surpassed the configured threshold, resulting in access denial.

The specific security rule that caused the denial is identified by its ID 949110 from the OWASP ModSecurity Core Rule Set (CRS), version 4.5.0-dev. The log further provides details on the request method and URI that were blocked. For example, the blocked request was a GET requestto the URI /bin/bash, which ModSecurity identified as suspicious.

Throughout the log, similar patterns can be observed. Multiple attempts from the same client (10.0.2.15) to access different URIs such as /WEB-INF/web.xml, /WEB-INF/Context.xml, and /WEB-INF/classes/ were all blocked by ModSecurity. Each attempt is logged with similar details, highlighting the consistent enforcement of security rulesby the WAF. Additionally, there are references to POST requests attempting to exploit PHP fileinclusion vulnerabilities, which were also blocked.

Overall, the log entries provide a comprehensive overview of the security measures in place, showcasing how ModSecurity effectively identifies and mitigates potential threats by blocking suspicious requests based on predefined rules and anomaly scoring mechanisms. This log is crucial for security administrators to monitor and analyze unauthorized access attempts and refine their security policies accordingly.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion:

Integrating the ModSecurity Web Application Firewall (WAF) with the Emerging Threats threat intelligence feed represents a significant milestone in enhancing the security framework of a web application. By carefully configuring ModSecurity and integrating threat intelligence rules, the defenses against a wide range of cyber threats have been significantly strengthened. The rigorous testing and deployment phases have validated the effectiveness of these security measures. Continuous monitoring of WAF logs and extensive testing of simulated attack scenarios have fine-tuned the WAF's ability to detect and mitigate potential threats swiftly. These efforts have not only reinforced the defenses but also ensured the reliability and resilience of the security infrastructure.

Additionally, the integration has enabled proactive identification and response to emerging threats, providing a robust and dynamic security posture. This comprehensive approach to security has established a high level of confidence in the protective measures implemented.

7.2 Future Scope:

Looking ahead, several strategic avenues for further advancement and expansion have been identified. These include integrating advanced threat detection technologies such as machine learning and artificial intelligence to enhance the WAF's ability to detect and respond to sophisticated cyber threats. Enhancing threat intelligence integration by expanding beyond Emerging Threats and continuously refining rule sets is also a priority. Additionally, implementing comprehensive cybersecurity training programs to educate users on best practices for secure behavior is essential. Integrating complementary security technologies like IDS, EPP, and SIEM systems will provide layered defenses and holistic security visibility.

Collaboration with industry peers and security communities is crucial for exchanging insights and emerging threat information, thereby enhancing proactive defense strategies. Adhering to regulatory compliance and governance standards through regular audits ensures that security measures align with legal frameworks. Embracing emerging technologies, fostering a cybersecurity culture, and leveraging industry partnerships are essential for sustaining robust protection, adapting to evolving threats, and effectively safeguarding digital assets.

References

[1] A. Tekerek, C. Gemci and O. F. Bay, 2014, "Development of a hybrid web application firewall to prevent web-based attacks", IEEE 8th International Conference on Application of Information and Communication Technologies (AICT), Astana, Kazakhstan, pp. 1-4, doi: 10.1109/ICAICT.2014.7035910.

[2] H. K. J and G. P. J 2023, Securing Web Application using Web Application Firewall (WAF) and Machine Learning. First International Conference on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI), Tiruchengode, India, pp. 1-8.

[3] T. Rahmawati, R. W. Shiddiq, M. R. Sumpena, S. Setiawan, N. Karna and S. N. Hertiana, 2023, "Web Application Firewall Using Proxy and Security Information and Event Management (SIEM) for OWASP Cyber Attack Detection", *IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, Bali, Indonesia, 2023, pp. 280-285, doi: 10.1109/IoTaIS60147.2023.10346051.

[4] K. R. Khamdamovich and I. Aziz, 2021, "Web application firewall method for detecting network attacks," *International Conference on Information Science and Communications Technologies (ICISCT)*, Tashkent, Uzbekistan, 2021, pp. 01-03, doi: 10.1109/ICISCT 52966.2021.9670285.

[5] R. A. Muzaki, O. C. Briliyant, M. A. Hasditama and H. Ritchi, 2020, "Improving Security of Web-Based Application Using ModSecurity and Reverse Proxy in Web Application Firewall," *2020 International Workshop on Big Data and Information Security (IWBIS)*, Depok, Indonesia, 2020, pp. 85-90, doi: 10.1109/IWBIS50925.2020.9255601.

[6] S. Wang, R. Liu, X. Guo and G. Wei, 2022, "Design of Web Application Firewall System through Convolutional Neural Network and Deep Learning," *International Conference on Computers, Information Processing and Advanced Education (CIPAE)*, Ottawa, ON, Canada, 2022, pp. 454-457, doi: 10.1109/CIPAE55637.2022.00101.

[7] V. Clincy and H. Shahriar, 2018, "Web Application Firewall: Network Security Models and Configuration," *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, 2018, pp. 835-836, doi: 10.1109/COMPSAC.2018.00144.

[8] M. Srokosz, D. Rusinek and B. Ksiezopolski, [2018], "A New WAF-Based Architecture for Protecting Web Applications Against CSRF Attacks in Malicious Environment," *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Poznan, Poland, pp. 391-395.

[9] L. Zhang and M. Huang, 2015, "A Firewall Rules Optimized Model Based on Service-Grouping", *12th Web Information System and Application Conference (WISA)*, Jinan, China, pp. 142-146, doi: 10.1109/WISA.2015.47.

[10] Z. Ghanbari, Y. Rahmani, H. Ghaffarian and M. H. Ahmadzadegan, 2015, "Comparative approach to web application firewalls", *2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Tehran, Iran, pp. 808-812, doi: 10.1109/KBEI.2015.7436148.

[11] Firkhan Ali Bin Hamid Ali, 2011, "A study of technology in firewall system", *IEEE Symposium on Business, Engineering and Industrial Applications (ISBEIA)*, Langkawi, Malaysia, pp. 232-236, doi: 10.1109/ISBEIA.2011.6088813.

[12] H. Takahashi, H. F. Ahmad and K. Mori, 2011, "Application for Autonomous Decentralized Multi Layer Cache System to Web Application Firewall," *Tenth International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, pp. 113-120, doi: 10.1109/ISADS.2011.20.

[13] A. El-Atawy, E. Al-Shaer, T. Tran and R. Boutaba, 2009, "Adaptive Early Packet Filtering for Defending Firewalls Against DoS Attacks", *IEEE INFOCOM 2009*, Rio de Janeiro, Brazil, 2009, pp. 2437-2445, doi: 10.1109/INFCOM.2009.5062171.

[14] F. Zhao, X. Peng and W. Zhao, "Multi-Tier Security Feature Modeling for Service-Oriented Application Integration," *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, Shanghai, China, 2009, pp. 1178-1183, doi: 10.1109/ICIS.2009.80.

[15] A. El-Atawy, E. Al-Shaer, T. Tran and R. Boutaba, "Adaptive Early Packet Filtering for Defending Firewalls Against DoS Attacks," *IEEE INFOCOM 2009*, Rio de Janeiro, Brazil,2009, pp. 2437-2445, doi: 10.1109/INFCOM.2009.5062171.

[16] G. Namit, S. Abakash, S. Dheeraj “Web Application Firewall”. CS499: B. Tech Project Final Report, 2008.

[17] W. Wang and J. Li, "An XML Firewall on Embedded Network Processor," *Fourth International Conference on Networking and Services (icns 2008),* Gosier, France, 2008, pp. 1-6, doi: 10.1109/ICNS.2008.15.

[18] Hamed H, Al-shaer E. Dynamic rule-ordering optimization for high-speed firewall filtering[J]. In Asiaccs 06: Acm Symposium on Information, 2006:332--342.

[19] Firkhan Ali, H. A., “ An Analysis of Possible Exploits in the Computer Network’s Security“ in ISC 2005 : Proceedings of the International Science Congress 2005. PWTC, Kuala Lumpur, 2005. pp.338.

[20] Yuan L, Chen H. FIREMAN: a toolkit for firewall modeling and analysis[C]. //IEEE Symposium on Security & Privacy. IEEE, 2006:15 pp. - 213.