future internet

MDPI

Article

Toward an SDN-Based Web Application Firewall: Defending against SQL Injection Attacks

Fahad M. Alotaibi [1,2] and Vassilios G. Vassilakis [3,*]

[1] Department of Computing, Imperial College London, London SW7 2BX, UK
[2] College of Science and Arts Sharoura, Najran University, Najran 66446, Saudi Arabia
[3] Department of Computer Science, University of York, York YO10 5GH, UK
* Correspondence: vasileios.vasilakis@york.ac.uk

Abstract: Web attacks pose a significant threat to enterprises, as attackers often target web applications first. Various solutions have been proposed to mitigate and reduce the severity of these threats, such as web application firewalls (WAFs). On the other hand, software-defined networking (SDN) technology has significantly improved network management and operation by providing centralized control for network administrators. In this work, we investigated the possibility of using SDN to implement a firewall capable of detecting and blocking web attacks. As a proof of concept, we designed and implemented a WAF to detect a known web attack, specifically SQL injection. Our design utilized two detection methods: signatures and regular expressions. The experimental results demonstrate that the SDN controller can successfully function as a WAF and detect SQL injection attacks. Furthermore, we implemented and compared ModSecurity, a traditional WAF, with our proposed SDN-based WAF. The results reveal that our system is more efficient in terms of TCP ACK latency, while ModSecurity exhibits a slightly lower overhead on the controller.

Keywords: web application firewall; software-defined network; network security

Citation: Alotaibi, F.M.; Vassilakis, V.G. Toward an SDN-Based Web Application Firewall: Defending against SQL Injection Attacks. *Future Internet* **2023**, *15*, 170. https://doi.org/10.3390/fi15050170

Academic Editor: Dimitrios Papakostas, Dimitrios Katsaros, Claude Chaudet

Received: 2 March 2023
Revised: 10 April 2023
Accepted: 28 April 2023
Published: 29 April 2023

1. Introduction

Web applications play a crucial role in today's digital age, providing users with easy access to information and enabling organizations to reach a broader audience. Consequently, the number of websites has grown exponentially over the past decade, increasing from hundreds of millions to nearly 1.7 billion [1,2]. However, this rapid expansion has also attracted adversaries seeking unauthorized access to organizational and user data. One persistent and significant threat is SQL injection, which ranks among the top 10 risks to web applications according to The Open Web Application Security Project (OWASP) [3]. Despite significant advancements in detection and defense technology, these threats persist due to inadequate user input validation

Although detection and defense technology has significantly evolved over time, many of the same threats persist. One primary cause of these threats is the lack of proper user input validation. Given the importance of web applications, various security solutions have been introduced to reduce the risk of cyberattacks at the application layer through monitoring and detection. One of the most prominent solutions is the use of web application firewalls (WAFs) such as ModSecurity [4] and FortiWeb [5]. In order to detect and block web-based attacks, WAFs might rely on different methods, such as attack signatures and regular expressions. Signature-based detection involves checking traffic against a database of accurate signatures for known attack patterns, whereas regular-expression-based detection uses a series of characters as a baseline for detecting multiple signatures instead of defining each pattern individually. These detection methods typically require full access to traffic and may utilize deep packet inspection (DPI).

The concept of software-defined networking (SDN) is based on removing decision making from switches and transferring it to a logically centralized controller [6]. SDN can

help to reduce operational and management costs, particularly the costs of configuring each network device in large networks. Specifically, SDN can be used to enhance network security by providing the centralized visibility, control, and management of security policies. It enables a dynamic enforcement of security policies based on network traffic patterns, application types, and user behaviors, improving threat detection and response capabilities [7]. The OpenFlow protocol facilitates communication between SDN controllers and switches by granting the controller access to the data plane [8,9].

However, the application of traditional security solutions to protect web applications or web servers using SDN has not been thoroughly explored in the literature. Existing research primarily focuses on improving malware/ransomware detection [10,11], implementing firewalls to protect networks from external attacks such as DDoS attacks [12–15], designing flexible honeynets [16], and developing generic moving target defense (MTD) systems [17,18]. Therefore, the design and implementation of an SDN-based WAF is a significant objective for establishing a primary security architecture for future SDN deployments. This work aims to be a pioneering step toward designing an SDN-based WAF specifically focused on defending against SQL injection attacks. The developed prototype was also experimentally compared with ModSecurity, a popular WAF.

This work aims to address the crucial question of whether a controller can reach or exceed the efficiency of traditional WAFs in detecting SQL injection attacks. Therefore, we did not aim to design a more complex configuration of SDN switches. Moreover, by presenting custom SDN-based security solutions, we demonstrate the potential to protect against these attacks without relying on expensive corporate software and hardware.

The rest of the paper is organized as follows. Section 2 briefly presents background information on SQL injections, WAFs, and DPI. Section 3 reviews related works, discusses their limitations, and highlights the novelty of our work. Section 4 clarifies our motivation and contribution. Section 5 presents our conceptual design of an SDN-based WAF. Section 6 provides implementation details. Section 7 presents our experimental setup and validation results. Section 8 presents evaluation results for our solution and compares them with ModSecurity WAF. Sections 9 and 10 conclude the paper and discuss future research directions.

2. Background

In this section, we briefly present the background information on SQL injections, WAFs, and DPI.

2.1. SQL Injection

Websites often need to interact with users and store, retrieve, and modify data in databases. In order to achieve this, an intermediary language facilitating communication with the database can be employed. A popular example is the structured query language (SQL), which enables database modifications through specific queries embedded in the web application source code [19]. However, databases may contain confidential and private data, making them the primary target for adversaries. SQL injection vulnerabilities can occur when user input validation is not properly implemented, allowing adversaries to manipulate statements and gain access to sensitive data, bypassing existing authentication or authorization mechanisms. In some cases, this can result in the theft of admin credentials or privilege escalation. Figure 1 illustrates an example of an SQL injection attack. When the user input is not adequately filtered, adversaries can inject packets containing SQL commands via POST and GET requests, allowing them to obtain, delete, or modify confidential data in the database.

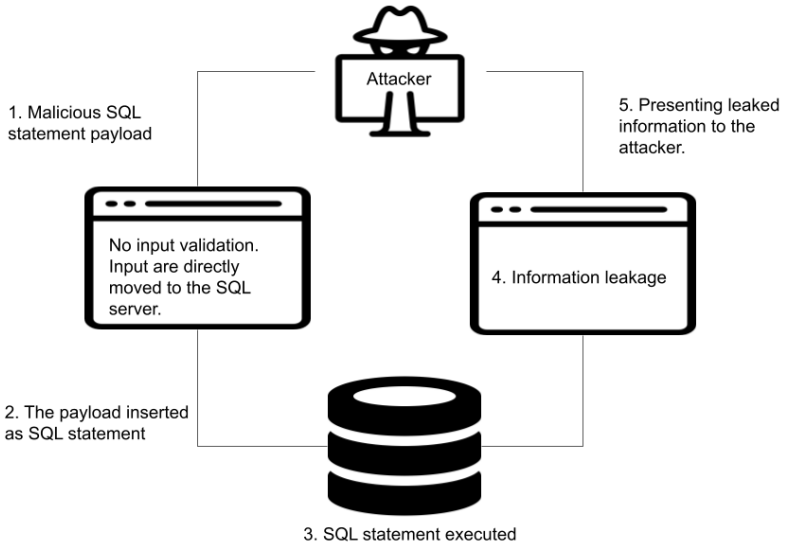

Figure 1. SQL injection attack mechanism.

2.2. Web Application Firewall

WAFs act as an intermediate layer of protection between users and web application servers [20]. The effectiveness of WAFs relies on the configured rules, as they inspect web traffic to detect patterns or texts specified in these rules. These rules typically contain malicious patterns or texts that are commonly used by attackers to exploit vulnerable services or known software vulnerabilities. Additionally, WAF rules define the action to be triggered upon a match, and decisions are made based on security policies. Traffic can be modified, blocked, or ignored, and a warning may be sent to the network administrator. Firewall rules can be based on blacklists (blocking traffic that matches the rules), whitelists (allowing only traffic that matches the rules), or a combination of both.

A typical workflow of a WAF includes the following stages. (i) Parsing requests/traffic: WAFs parse and normalize the incoming traffic to ensure a unified format, making further processing easier. (ii) Normalization: WAFs normalize the parsed data to eliminate inconsistencies and reduce the risk of evasion techniques used by attackers. (iii) Matching: WAFs perform matching based on their configured rules, which can be either signature-based or anomaly-based. These rules are designed to detect known patterns or anomalies that may indicate an attack. (iv) Logging: Upon a match, important information about the detected event is logged for further inspection by security staff. This helps in analyzing and investigating potential security incidents.

2.3. Deep Packet Inspection

DPI is a security technique that provides a more comprehensive data inspection compared to traditional firewall features, allowing for a thorough analysis of incoming packets [21]. DPI involves analyzing incoming packets in-depth, covering multiple layers within the packet (Figure 2). This approach allows for a thorough detection of policy violations, such as visiting unwanted websites or exploiting vulnerabilities. Detection rules are stored in databases, and these rules and corresponding actions are matched with each packet passing through the DPI checkpoint. DPI is commonly used in WAFs to block malicious content in HTTP traffic. However, DPI, being based on signatures, can be bypassed using advanced adversarial techniques. Additionally, DPI generates extra traffic in the network, which may result in delays or packet loss.

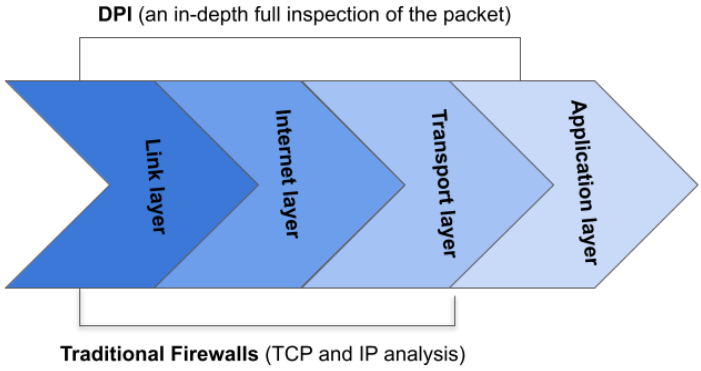

Figure 2. DPI architecture based on the TCP/IP model.

3. Related Work

In this section, we review related works that focus on utilizing SDN to implement firewalls against network attacks and security mechanisms using DPI. We also discuss recent research on regular expression implementation and the application of DPI to detect SQL injection attacks in non-SDN environments, such as traditional networks and cloud environments. Finally, we highlight the novelty of our work in SDN environments.

Although the literature does not extensively discuss implementing security solutions for web applications using SDN, there are several solutions that utilize features implemented in WAFs, such as DPI [22,23]. DPI involves inspecting the contents of network packets at the application layer to identify patterns or signatures of known attacks. SDN controllers can be programmed to use DPI techniques to analyze the payload of network packets and identify suspicious patterns that may indicate application-level attacks, such as SQL injection, cross-site scripting (XSS), or command injection attacks. Researchers have utilized SDN features to analyze both SMB and HTTP traffic for the purpose of detecting and blocking ransomware [22]. In order to enable DPI, the authors modified the POX controller to allow for full traffic forwarding from the OpenFlow switch to the controller. Other works have also explored the use of SDN for application layer firewalls, aiming to block hosts from accessing specific websites [24,25]. Additionally, network firewalls for SDN have been investigated by various researchers. Early efforts focused on implementing a layer 3 firewall that matches rules with packet headers [26]. Subsequently, researchers extended this work to implement stateless layer 2 [27], layer 1-4 [28], layer 2-4 [29], and stateful firewalls [30,31].

The authors of [32] proposed generating regular expression rules automatically using machine learning and multi-objective genetic algorithms to detect SQL injection attacks. In the cloud environment, the authors of [33] proposed a system that inspects traffic using regular expressions to raise awareness among cloud tenants and assess SQL injection attacks.

Although SDN security has been studied in the literature, there is a lack of explicit focus on implementing a WAF to block web attacks. The only existing works on layer 7 firewalls have aimed at preventing access to certain websites, without addressing web attacks directly [24,25]. Given the increasing prevalence and impact of web attacks, it is crucial to explore solutions for web attacks that consider the novelty of SDN and its growing adoption by organizations. Therefore, this work aims to contribute to the literature by implementing an SDN-based WAF to mitigate web attacks. Furthermore, our work shares two goals with [26]: to raise awareness among information security researchers about the potential of SDN as a new field and to explore the feasibility of relying on custom SDN-based security solutions without the need for proprietary software or hardware. However, our work differs from [26] in that we specifically aim to design a firewall to mitigate web attacks, whereas most previous works have focused on network-level attacks.

This distinction highlights the unique contribution of our work in addressing the specific challenges and requirements of web application security in an SDN environment.

In addition to DPI, there are other methods that can also be used in SDN environments to detect application-level attacks. Some of the commonly used methods include:

- Flow Monitoring: SDN controllers can monitor network flows, which are sequences of packets that belong to the same communication session or application transaction, in order to detect anomalies that may indicate application-level attacks [34]. Flow monitoring involves analyzing various flow attributes, such as the packet size, packet rate, inter-arrival time, and communication patterns, to detect deviations from normal behavior that may indicate attacks.
- Behavioral Analysis: SDN controllers can use behavioral analysis techniques to establish baseline behavior for applications or users and detect deviations from the baseline that may indicate attacks. For example, by monitoring application-level traffic patterns in vehicular ad hoc networks (VANETs) over time, an SDN controller can learn normal behavior and raise alerts when it detects anomalies, such as sudden spikes in traffic or unusual communication patterns [35].
- Machine Learning (ML): ML algorithms can be used in SDN environments to detect application-level attacks by learning from historical data and identifying patterns that may indicate attacks [10]. ML algorithms can analyze large volumes of network data, including packet payloads, flow attributes, and communication patterns, to detect unknown or zero-day attacks that may not have known signatures [36].
- Hybrid Approaches: SDN controllers can use a combination of the above methods in a hybrid approach to detect application-level attacks. By leveraging multiple techniques, such as DPI, flow monitoring, behavioral analysis, ML, and signature-based detection, SDN controllers can increase the accuracy and effectiveness of attack detection.

It is important to note that no single method is foolproof, and a combination of techniques may be needed to achieve robust application-level attack detection in SDN environments. The choice of methods to use will depend on the specific requirements, constraints, and capabilities of the SDN environment and the types of application-level attacks being targeted. The proper configuration, tuning, and monitoring of the chosen detection methods are also essential to minimize false positives and false negatives and ensure effective application-level attack detection in SDN environments.

4. Motivation and Contributions

In the previous sections, several issues have been raised regarding the use of SDN as a security solution without the need for third-party applications. Existing literature has primarily focused on limiting the risks of network attacks such as DDoS [15] and combatting malicious software attacks [23]. However, the possibility of using SDN to counter web attacks, particularly SQL injection attacks, has not been adequately addressed. Furthermore, the proven ability of SDN controllers to analyze traffic in previous works [22–25] can be leveraged to detect suspicious patterns and texts in web traffic. However, an important question arises: can an SDN controller achieve or surpass the efficiency of traditional WAFs? This motivates our work to explore the potential of SDN as a WAF and make contributions to the literature in this regard.

Specifically, this work aims to present an SDN-based solution, evaluate its capabilities, and explore research directions for SDN-based web security solutions. The contributions of this paper are as follows:

1. The design, implementation, and evaluation of an SDN-based WAF prototype.
2. The design and implementation of an SDN-WAF framework to counter SQL injection attacks.
3. The implementation of a traditional WAF (ModSecurity) to counter SQL injection attacks and a comparison with the SDN-based WAF.
4. An evaluation of the efficiency of the SDN-based WAF and the potential of using such solutions as alternatives to traditional WAFs.

Our proposed framework utilizes DPI to thoroughly analyze incoming traffic, taking into account the possibility of multiple attackers. By processing Packet_In messages even after routes are configured, we ensure the early detection and mitigation of SQL injection attacks.

5. Conceptual Design

In this section, we present the conceptual design of our WAF and explain the stages that the traffic passes through in the SDN. The implementation process of these stages is illustrated in Figure 3. Our WAF goes through the following stages:

1. *Reception stage:* The controller receives all traffic within the network from the data plane using the OpenFlow protocol, and filters the traffic of interest. This stage analyzes the sender/receiver address in layer 3 to identify the address of the web server. After that, it verifies if the port is for the HTTP protocol for either the sender or the receiver. If these conditions are met, the packet is moved to the next stage.
2. *Processing stage:* At this point, the packet is processed to convert it into a textual format. This stage standardizes the traffic that will be taken to the next stage for inspection and for searching for suspicious patterns.
3. *Inspection stage:* This stage is the primary function of the WAF, as it focuses on inspecting traffic for matching suspicious patterns. In order to achieve this, all traffic is transferred to the controller for inspection.
4. *Containment stage:* When a match occurs in the inspection stage, the process moves to the containment stage, where the suspicious packet is first dropped and then the IP address of the machine responsible for the packet is blocked. After that, the suspicious packet is recorded on the controller for future inspection and investigation by information security personnel.

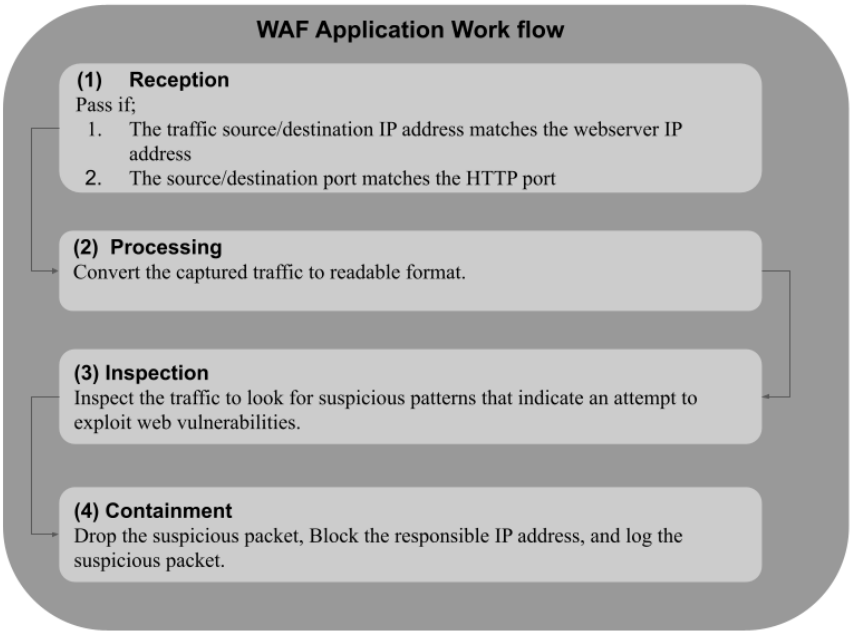

Figure 3. WAF conceptual design.

6. Implementation

In order to implement the design and approaches discussed in the previous section, we chose POX as the controller (running OpenFlow v1.0) and OpenvSwitch as the data plane element. In this section, we discuss the mechanism for applying the WAF stages to the controller and the proposed solutions for implementing a WAF capable of detecting SQL injection attacks.

6.1. Implementation Stages

As mentioned earlier, the WAF passes through four stages, and each stage has a specific task. The implementation mechanisms for each stage at the POX controller are as follows:

1. *Reception stage:* In order to receive the full traffic, we need to modify the POX controller. The modifications include:
   - Using the misc.full_forwarding function in POX controller, which allows the controller to receive all traffic that does not match any existing flow entries.
   - Modifying the libopenflow_01.py file in the POX/OpenvSwitch folder to configure the behavior of OpenFlow messages.

   Furthermore, in the reception stage, the destination IP address of the incoming packets is matched with the IP address of the web server. If there is a match, the traffic is forwarded to the next stage for further processing.
2. *Processing stage:* In this stage, the pack() function in Python is used to convert the in-wire data to strings, making it possible for the data to be inspected in the next stage. Additionally, the entire traffic is converted to lowercase to avoid simple evasion techniques. It is worth noting that while Python may not be the most efficient language for processing high-speed traffic; it was chosen for its ease of use and versatility in this implementation. The processing stage in the SDN-based WAF is not expected to be the bottleneck for performance, and any delay added by using Python for traffic handling is likely to be negligible. However, in cases where efficiency is crucial during the processing stage, alternative solutions may need to be considered, such as using other programming languages or optimizing the Python code for performance. It is important to thoroughly test and validate the implementation to ensure it meets the performance requirements of the specific use case.
3. *Inspection stage:* The inspection stage of the WAF is responsible for inspecting the traffic for suspicious patterns, particularly SQL injection attacks. There are two main applications implemented in this stage:
   (a) Matching the content of the packet with predefined signatures indicating the existence of an SQL injection exploitation: In this approach, known signatures are used to detect malicious GET and POST requests. These signatures are predefined and can indicate the presence of SQL injection attacks in the traffic. This approach can detect both GET and POST requests, as all of the traffic is inspected. Table 1 shows an example of known signatures that can be used for SQL injection detection.
   (b) Matching with regular expressions for whitelisting or blacklisting: In this approach, regular expressions are used to match the content of the packet against a set of predefined patterns, which can be used for whitelisting or blacklisting purposes. In the implementation described, whitelisting was used for custom solutions that fit the installed vulnerable web application. This allows for designing customized solutions that suit the environment/network. As a result, when an adversary sends an anomalous HTTP GET request, the application will classify this request as malicious.

   It is important to keep the signatures and regular expressions up-to-date, as attackers continuously evolve their techniques. Regular updates and monitoring of the WAF rules are necessary to ensure its effectiveness in detecting and mitigating SQL injection attacks. Additionally, it is essential to thoroughly test and validate the implementation with real-world traffic to ensure its accuracy and reliability.
4. *Containment stage:*
   The containment stage of the WAF is responsible for taking action against detected SQL injection attacks. When a match occurs, the following steps are taken:
   (a) Blocking the IP address of the party responsible for the exploitation: A new rule is added to the OpenvSwitch to prevent the suspicious party from sending any

further traffic to the web server. This is typically carried out by blocking the IP address of the detected attacker. However, in cases where the attacker is behind a network address translation (NAT) device, blocking the NAT-configured IP may potentially affect legitimate users. In such cases, alternative solutions can be considered, such as blocking the MAC address or providing both the user's IP and the NAT IP for more precise blocking.

(b) Dropping the suspicious packet: The packet that triggered the SQL injection detection is dropped using the halt() function, which prevents it from reaching the web server. This effectively contains the attack and prevents further damage.

(c) Logging the suspicious packet for future investigation: The dropped packet is recorded in the logs folder for further investigation and analysis. This can help in understanding the nature of the attack, identifying patterns, and improving the effectiveness of the WAF in detecting and mitigating SQL injection attacks.

It is important to note that false positives may occur, where legitimate traffic may trigger the SQL injection detection. In order to minimize the impact on legitimate users, specific ports are blocked instead of blocking all ports, as blocking an IP from using all services may have a greater negative impact. The continuous monitoring and fine-tuning of the containment measures are necessary to strike a balance between effectively mitigating attacks and minimizing false positives.

Table 1. Signatures used to detect SQL injection.

order%20by	order by	order+by	union all
union+all	union%20all	union%20all	1,2,3
union+select	union%20select	union select	version()

6.2. SQL Injection Detection

In this subsection, we provide a more in-depth explanation of the modules used in the inspection stage for SQL injection detection. These modules rely on signatures and regular expressions.

6.2.1. Module 1: Signature-Based

This module aims to demonstrate the usability of signature-based detection systems within the scope of the SDN, without the need for intermediate software or hardware provided by vendors. Several common signatures were selected for detecting SQL injection attacks, as shown in Table 1. This module relies on storing the signatures in a list and implementing the first two stages, namely reception and processing. Subsequently, these signatures are passed into a for loop to match them with the traffic directed to the web server. Algorithm 1 summarizes the mechanism of this module. Note that, in practice, web servers can run on non-standard ports. In our work, we considered the standard port 80 for the HTTP protocol. This assumption was made for simplicity and to align with common practices.

6.2.2. Module 2: Regular-Expression-Based

Regular expressions are commonly used to detect attacks against web applications. Thus, the possibility of using them in SDN was studied. In this module, we implemented regular expressions to scan traffic for GET requests and then extract the URL path. If no path is requested, the traffic will be forwarded to the web server. For example, a path can be defined as \filename.php?id=[0-9]. If this path exists in the request, the application will move to the next stage, which compares the path with the predefined path extracted from our analysis of HTTP GET requests in SDN. The path should be \filename.php?id=[0-9] http. If there are changes in this path, the request is considered malicious. Algorithm 2 shows the design of this module and the regular expression used for detection. Note that

the example provided is just one possible case; in our work, we thoroughly examined all possible paths.

Algorithm 1 Signature-based algorithm

Input: PacketIn event
Output: None
Function: Signature-based algorithm

```
packet ← PacketIn.tcp
signatures ← "order+by", "union select", "union+select", "order by", etc.
if packet is None then
    return
else if packet.dstport == 80 then
    for i in signatures do
        if packet.find(i) is True then
            Install new entry to block the sender IP address from sending from port 80
            PacketIn.halt
        else
            return
        end if
    end for
else
    return
end if
```

Algorithm 2 Regular-expression-based algorithm

Input: PacketIn event
Output: None
Function: Regular-expression-based algorithm

```
packet ← PacketIn.tcp
FirstReg ← re.compile("([\w\.-]+)\?([\w\.-]+)\=([\w\.-]+) \s*")
SecondReg ← re.compile("([\w\.-]+)\?([\w\.-]+)\=([\w\.-]+) \s*(http+)")
if packet is None then
    return
else if packet.dstport == 80 then
    if FirstReg.search(packet) then
        if SecondReg.search(packet) is None then
            Install new entry to block the sender IP address from sending from port 80
            PacketIn.halt
        else
            return
        end if
    else
        return
    end if
else
    return
end if
```

The proposed SDN-based web firewall can efficiently scale to detect multiple types of attacks, such as cross-site scripting (XSS) and local file inclusion (LFI), by leveraging both signature-based and regular expression components when processing traffic directed toward the web server. Additionally, incorporating statistical features, such as the number of exchanged packets, could further enhance the approach to address various attack types.

7. Experimental Setup

In this section, the lab used and the configuration used to make it compatible with the proposed modules are explained. Furthermore, the controller environment that hosts the solutions and how the solutions are implemented on it are explained. Finally, the effectiveness of the proposed solutions against SQL injection attacks is verified. This section aims to answer whether SDN can act as a WAF without the need for traditional WAFs. Additionally, our implementation of a traditional WAF was explained and compared experimentally with the SDN-based WAF prototype.

7.1. Lab Setup

The lab was configured in a virtual environment using VirtualBox (Oracle Corporation, Austin, TX, USA). The specifications of the machine used are as follows: Lenovo IdeaPad 530S Ultrabook Core i7 8550U, 1.80 GHz with 16 GB RAM. The main system installed on the machine is Ubuntu, which functions as an OpenvSwitch and also as a host for the virtual environment. In order to configure Ubuntu to run as an OpenvSwitch 2.7.0, the kernel was downgraded to 4.4.0. The systems in the virtual environment are as follows: Ubuntu Xenial as the POX controller, Debian as a web server containing the vulnerable web application, and Ubuntu Focal with penetration testing tools, specifically SQLMap. Figure 4 shows the details of the lab topology and configuration.

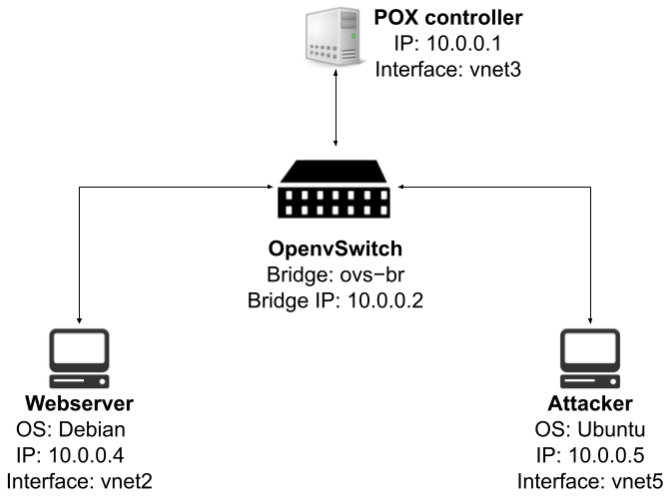

Figure 4. Installed SDN testbed topology and configuration.

7.2. Modules Implementation

In our implementation of the two modules, we relied on two functions in POX, specifically the PackitIn function, which determines the arrival of a new packet to the controller, and the launch function, which runs the application. The code for the two developed modules can be found on GitHub [37]. In order to run the applications in the controller, we used the same command as both applications need to access all of the forwarded traffic, so changes in the reception stage must be implemented. Therefore, in order to launch the modules, we needed to trigger the following command: ./pox.py misc.full_payload forwarding.l2_learning Signatures Regular. This command runs both applications. However, for a more verbose and informative interface, the command ./pox.py misc.full_payload forwarding.l2_learning Signatures Regular samples.pretty_log log.level—DEBUG info.packet_dump can be used. This will show notifications and actions that have been raised by the controller.

7.3. Modules Validation

SQLmap was used on the attacker's machine to test and evaluate the effectiveness of the implemented modules. For greater accuracy, we started each module separately in the POX controller, then launched the attack, and finally triggered the detection mechanism. In the signature-based application, the attack was detected directly, and the SQLmap tool was prevented from exploiting the SQL injection vulnerability, as shown in Figure 5. In the regular-expression-based application, all SQLmap attacks were also detected, as shown in Figure 6. Moreover, in the containment stage, all of the payloads were recorded and saved in the logs folder, as shown in Figure 7. An example of one of these payloads is shown in Figure 8. Therefore, the effectiveness of SDN in detecting known web attacks and threats has been demonstrated using two widely used methods for this purpose.

Figure 5. Detect SQLmap using signatures.

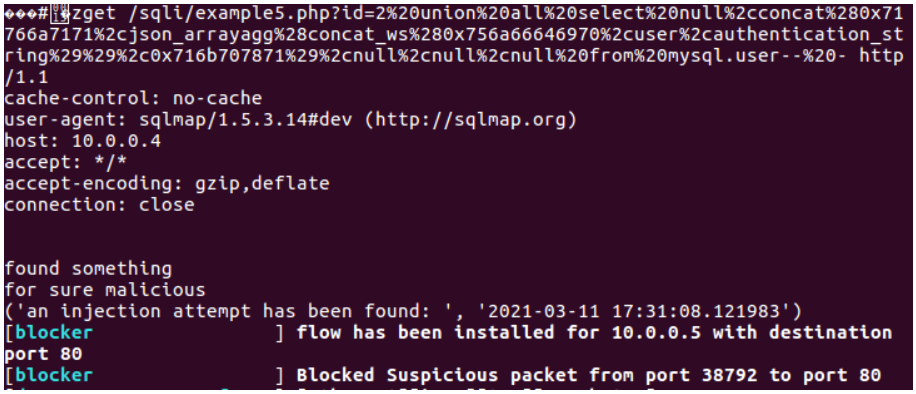

Figure 6. Detect SQLmap using regular expressions.

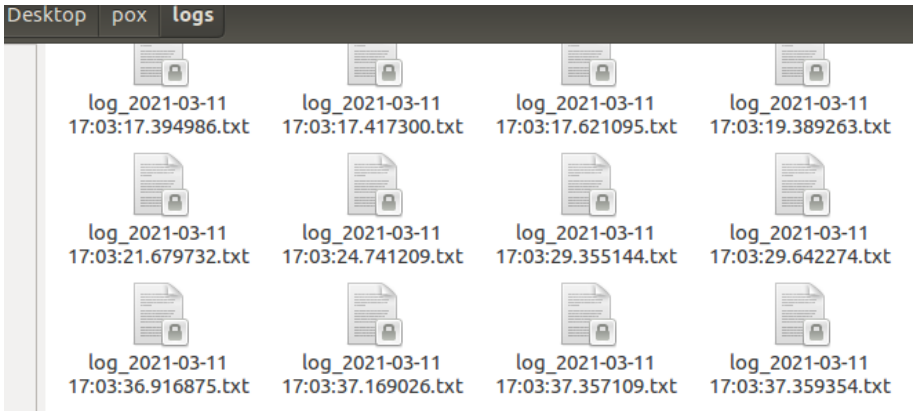

Figure 7. List of detected attacks.

Figure 8. Detected payload record.

7.4. ModSecurity WAF Implementation

A ModSecurity WAF was deployed on the web server machine, and the OWASP ModSecurity core rule set (CRS) was used. The forwarding.l2_learning function was executed on the controller, which turns the controller into a layer 2 learning switch. ModSecurity was able to block all SQLmap attempts to inject vulnerable application, as all requests made by SQLmap have been forbidden.

8. Performance Evaluation and Comparison

Signature-based WAFs can efficiently detect and prevent security threats, but their effectiveness depends on the accuracy of the rules or signatures in the database. Therefore, the detection capability we have previously defined for both traditional WAFs and SDN-based WAFs is just an indication of the detection possibility and should not be considered as a measure of comparison. In order to determine which approach is more efficient for SDN-based WAFs, we conducted performance measurements in terms of TCP ACK latency and CPU usage overhead (effect on the controller's efficiency).

We measured the time it takes to send ACK packets to port 80 of the web server and calculated the average latency over a ten-minute period. We also measured the average CPU usage of the controller over the same time period. Our results show that the developed SDN-based modules outperform ModSecurity in terms of TCP ACK latency. This means that packets in traditional WAFs require longer processing times, as they pass through the controller for processing and routing, then go to the WAF for inspection, and finally to the web server for request processing. On the other hand, in the SDN-based WAF, the stages of determining the route and inspection are combined in the controller, resulting in a faster packet arrival. However, ModSecurity performed better in terms of the CPU usage overhead on the controller, as the controller only directs and does not inspect them. Therefore, it can be concluded that SDN-based WAFs are better when the packet transfer speed is of importance to the network administrator, but traditional WAFs are better when the processing capacity of the controller is a priority. Table 2 summarizes these evaluation results.

Table 2. Evaluation results.

Module	TCP ACK Latency (Avg)	CPU Usage (Avg)
Traditional WAF (ModSecurity)	50.6 ms	7.02 %
SDN-based WAF (signatures)	43.6 ms	14.78 %
SDN-based WAF (regular expression)	35.5 ms	7.046 %

Note that detecting SQL injection attacks after route establishment necessitates forwarding all packets to the controller, introducing notable overhead. However, as demonstrated earlier, the overhead is less than that incurred by using a third-party WAF, making it acceptable in some scenarios. Additionally, although controller vulnerability is a valid concern [38,39], it is beyond the scope of our paper and merits further exploration.

9. Conclusions

This paper has presented solutions for detecting web attacks using SDN. Two modules were designed and implemented to detect web attacks, specifically utilizing signatures and regular expressions. These modules were evaluated and several important questions were addressed: can SDN be used as an effective and efficient WAF? Can traditional web attack solutions be implemented without modifying the SDN? How should we choose between an SDN-based WAF and traditional WAFs? These questions were answered and the most notable contribution was the demonstration of the concept using SQL injection attacks. Initially, an SDN-based WAF was designed, its stages were discussed, and it was implemented on the SDN. Several modifications were made to the POX controller to enable deep packet inspection. Then, three modules were implemented to detect SQL injection attacks: signature-based, regular-expression-based, and traditional WAFs using ModSecurity. The experimental results show that all of these solutions were able to detect infected web application injection attempts.

This work aimed to determine if a controller can achieve or surpass the efficacy of traditional WAFs in detecting SQL injection attacks using a simple SDN switch configuration. Furthermore, by showcasing custom SDN-based security solutions, we highlight the possibility of defending against these attacks without relying on costly corporate software and hardware.

In addition, other important aspects were measured to assist network managers in selecting the most appropriate solution. Specifically, the packet latency and CPU overhead were evaluated. The proposed solutions were better in terms of TCP latency, with average packet transmission times of 35.5 ms for regular expressions, 43.6 ms for signatures, and 50.6 ms for the traditional solution. This is due to the packet routing and processing processes, where, in the ModSecurity implementation, the packet is transferred to the controller for routing, then to the firewall for inspection, and finally to the web application for request processing. On the other hand, ModSecurity showed a lower CPU overhead, with an average CPU usage of 7.02%, compared to 7.046% for regular expressions and 14.78% for signatures.

10. Future Work

In future work, our aim is to enhance the reliability of the SQL injection attack detection system by incorporating a larger number of signatures and regular expressions. We also plan to study bypassing techniques employed against WAFs and develop effective anti-bypassing solutions based on SDN. Furthermore, we intend to test these solutions in larger networks to measure their impact on the controller and throughput. Additionally, we plan to explore and implement other solutions to mitigate SQL injection attacks and evaluate their effectiveness. Moreover, this work serves as a foundation for further research in web security using SDN, where solutions can be developed to detect data leakage, advanced web attacks, and automated attacks, as well as provide countermeasures against firewall-bypassing methods.

Lastly, the scalability of our proposed SDN-based web firewall framework needs to be thoroughly studied, particularly in real-world networks with multiple switches and potential attacks from the Internet. Our approach processes traffic regardless of its origin, and although we did not specifically address NAT in our current implementation, it serves as a starting point. Future work could enhance the framework by creating a distributed architecture to manage multiple switches and employing adaptive strategies for diverse network topologies, ensuring robust protection against SQL injection attacks in complex scenarios.

Author Contributions: Conceptualization, F.M.A. and V.G.V.; methodology, F.M.A. and V.G.V.; software, F.M.A.; validation, F.M.A.; formal analysis, F.M.A.; writing—original draft preparation, F.M.A.; writing—review and editing, V.G.V.; visualization, F.M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central Processing Unit
CRS	Core Rule Set
DPI	Deep Packet Inspection
HTTP	Hypertext Transfer Protocol
LFI	Local File Inclusion
ML	Machine Learning
MTD	Moving Target Defence
NAT	Network Address Translation
RAM	Random Access Memory
SDN	Software-Defined Networking
SQL	Structured Query Language
TCP	Transmission Control Protocol
WAF	Web Application Firewall
XSS	Cross-Site Scripting

References

1. Armstrong, M. How Many Websites Are There? August. Available online: https://www.statista.com/chart/19058/how-many-websites-are-there/ (accessed on 25 February 2023).
2. NetCraft. October 2022 Web Server Survey. October. Available online: https://news.netcraft.com/archives/category/web-server-survey/ (accessed on 25 February 2023).
3. OWASP. Top Ten Web Application Security Risks. Available online: https://owasp.org/www-project-top-ten/ (accessed on 25 February 2023).
4. SpiderLabs. ModSecurity. Available online: https://github.com/SpiderLabs/ModSecurity (accessed on 25 February 2023).
5. FortiWeb. Web Application Firewall (WAF). Available online: https://www.fortinet.com/products/web-application-firewall/fortiweb (accessed on 25 February 2023).
6. Kreutz, D.; Ramos, F.M.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
7. Girdler, T.; Vassilakis, V.G. Implementing an intrusion detection and prevention system using Software-Defined Networking: Defending against ARP spoofing attacks and Blacklisted MAC Addresses. *Comput. Electr. Eng.* **2021**, *90*, 106990. [CrossRef]
8. Lara, A.; Kolasani, A.; Ramamurthy, B. Network innovation using OpenFlow: A survey. *IEEE Commun. Surv. Tutorials* **2013**, *16*, 493–512. [CrossRef]
9. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
10. Cusack, G.; Michel, O.; Keller, E. Machine learning-based detection of ransomware using SDN. In Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, Tempe, AZ, USA, 21 March 2018; pp. 1–6.
11. Akbanov, M.; Vassilakis, V.G.; Logothetis, M.D. Ransomware detection and mitigation using software-defined networking: The case of WannaCry. *Comput. Electr. Eng.* **2019**, *76*, 111–121. [CrossRef]
12. Chica, J.C.C.; Imbachi, J.C.; Vega, J.F.B. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. [CrossRef]
13. Birkinshaw, C.; Rouka, E.; Vassilakis, V.G. Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks. *J. Netw. Comput. Appl.* **2019**, *136*, 71–85. [CrossRef]
14. Balarezo, J.F.; Wang, S.; Chavez, K.G.; Al-Hourani, A.; Kandeepan, S. A survey on DoS/DDoS attacks mathematical modelling for traditional, SDN and virtual networks. *Eng. Sci. Technol.* **2022**, *31*, 101065. [CrossRef]
15. Bawany, N.Z.; Shamsi, J.A.; Salah, K. DDoS attack detection and mitigation using SDN: Methods, practices, and solutions. *Arab. J. Sci. Eng.* **2017**, *42*, 425–441. [CrossRef]
16. Wang, H.; Wu, B. SDN-based hybrid honeypot for attack capture. In Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 15–17 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1602–1606.

17. Kim, M.; Cho, J.H.; Lim, H.; Moore, T.J.; Nelson, F.F.; Kim, D.D. Performance and Security Evaluation of a Moving Target Defense Based on a Software-Defined Networking Environment. In Proceedings of the 2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC), Beijing, China, 28 November–1 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 119–129.
18. Nguyen, T.A.; Kim, M.; Lee, J.; Min, D.; Lee, J.W.; Kim, D. Performability evaluation of switch-over Moving Target Defence mechanisms in a Software Defined Networking using stochastic reward nets. *J. Netw. Comput. Appl.* **2022**, *199*, 103267. [CrossRef]
19. Clarke, J. *SQL Injection Attacks and Defense*; Elsevier: Amsterdam, The Netherlands, 2009.
20. Clincy, V.; Shahriar, H. Web application firewall: Network security models and configuration. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japa, 23–27 July 2018; IEEE: Piscataway, NJ, USA, 2018; Volume 1, pp. 835–836.
21. Antonello, R.; Fernandes, S.; Kamienski, C.; Sadok, D.; Kelner, J.; Godor, I.; Szabo, G.; Westholm, T. Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. *J. Netw. Comput. Appl.* **2012**, *35*, 1863–1878. [CrossRef]
22. Rouka, E.; Birkinshaw, C.; Vassilakis, V.G. SDN-based Malware Detection and Mitigation: The Case of ExPetr Ransomware. In Proceedings of the 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2–5 February 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 150–155.
23. Alotaibi, F.M.; Vassilakis, V.G. SDN-Based Detection of Self-Propagating Ransomware: The Case of BadRabbit. *IEEE Access* **2021**, *9*, 28039–28058. [CrossRef]
24. Shieha, A. Application Layer Firewall Using OpenFlow. Master's Thesis, University of Colorado Boulder: Boulder , CO, USA, 2014.
25. Badotra, S.; Singh, J. Creating firewall in transport layer and application layer using software defined networking. In *Innovations in Computer Science and Engineering*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 95–103.
26. Suh, M.; Park, S.H.; Lee, B.; Yang, S. Building firewall over the software-defined network controller. In Proceedings of the 16th International Conference on Advanced Communication Technology, Pyeongchang, Republic of Korea, 16–19 February 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 744–748.
27. Javid, T.; Riaz, T.; Rasheed, A. A layer2 firewall for software defined network. In Proceedings of the 2014 Conference on Information Assurance and Cyber Security (CIACS), Rawalpindi, Pakistan, 12–13 June 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 39–42.
28. Kaur, K.; Kumar, K.; Singh, J.; Ghumman, N.S. Programmable firewall using software defined networking. In Proceedings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE: Piscataway, NJ, USA, 2015; pp. 2125–2129.
29. Othman, W.M.; Chen, H.; Al-Moalmi, A.; Hadi, A.N. Implementation and performance analysis of SDN firewall on POX controller. In Proceedings of the 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), Guangzhou, China, 6–8 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1461–1466.
30. Caprolu, M.; Raponi, S.; Di Pietro, R. Fortress: An efficient and distributed firewall for stateful data plane SDN. *Secur. Commun. Networks* **2019**, *2019*, 6874592. [CrossRef]
31. Kaur, K.; Singh, J. Building stateful firewall over software defined networking. In *Information Systems Design and Intelligent Applications*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 159–168.
32. Appelt, D.; Panichella, A.; Briand, L. Automatically repairing web application firewalls based on successful SQL injection attacks. In Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France, 23–26 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 339–350.
33. Gu, H.; Zhang, J.; Liu, T.; Hu, M.; Zhou, J.; Wei, T.; Chen, M. DIAVA: A traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data. *IEEE Trans. Reliab.* **2019**, *69*, 188–202. [CrossRef]
34. Yang, Z.; Yeung, K.L. Flow monitoring scheme design in SDN. *Comput. Networks* **2020**, *167*, 107007. [CrossRef]
35. Bhatia, J.; Dave, R.; Bhayani, H.; Tanwar, S.; Nayyar, A. SDN-based real-time urban traffic analysis in VANET environment. *Comput. Commun.* **2020**, *149*, 162–175. [CrossRef]
36. Tuan, N.N.; Hung, P.H.; Nghia, N.D.; Tho, N.V.; Phan, T.V.; Thanh, N.H. A DDoS attack mitigation scheme in ISP networks using machine learning based on SDN. *Electronics* **2020**, *9*, 413. [CrossRef]
37. Alotaibi, F.M. Falkarshmi/SDN-WAF. March. Available online: https://github.com/Falkarshmi/SDN-WAF (accessed on 26 February 2023).
38. Badotra, S.; Tanwar, S.; Bharany, S.; Rehman, A.U.; Eldin, E.T.; Ghamry, N.A.; Shafiq, M. A DDoS Vulnerability Analysis System against Distributed SDN Controllers in a Cloud Computing Environment. *Electronics* **2022**, *11*, 3120. [CrossRef]
39. Zhu, L.; Karim, M.M.; Sharif, K.; Xu, C.; Li, F.; Du, X.; Guizani, M. SDN controllers: A comprehensive analysis and performance evaluation study. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 1–40. [CrossRef]