applied sciences

MDPI

Article

Packet Injection Exploiting Attack and Mitigation in Software-Defined Networks

Jishuai Li, Sujuan Qin *, Tengfei Tu *, Hua Zhang and Yongsheng Li

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; sky_lee1990@bupt.edu.cn (J.L.); zhanghua_288@bupt.edu.cn (H.Z.); lee_yongsheng@bupt.edu.cn (Y.L.)

* Correspondence: qsujuan@bupt.edu.cn (S.Q.); tutengfei.kevin@bupt.edu.cn (T.T.)

Abstract: Software-defined networking (SDN) decouples the control plane and data plane through OpenFlow technology and allows flexible network control. It has been widely applied in different areas and has become a focus of attention in the future network. With SDN's development, its security problem has become a necessary point of research to be solved urgently. In this paper, we propose a novel attack, namely, the packet injection exploiting attack. By maliciously injecting false hosts into SDN network topology, attackers can further use them to launch a denial of service (DoS) attack. The consequences affect the throughput and processing capabilities of the controller, severely consume data plane resources, and ultimately affect the entire network. To prevent the packet-injection exploiting attack, we designed PIEDefender, an efficient, protocol-independent component built on SDN controllers to detect and mitigate attacks effectively. We implement the PIEDefender prototype on the Floodlight controller and assess the effectiveness in the software environment. Experimental results show that PIEDefender achieves a 97.8% injection detection precision and a 97.96% DoS detection precision, incurring an average CPU consumption of 10%. The evaluation demonstrates that the PIEDefender can effectively mitigate the attack against SDN with limited overhead.

Keywords: software-defined networking (SDN); OpenFlow; packet injection exploiting attack; false hosts; denial-of-service (DoS); detection; defense

Citation: Li, J.; Qin, S.; Tu, T.; Zhang, H.; Li, Y. Packet Injection Exploiting Attack and Mitigation in Software-Defined Networks. *Appl. Sci.* **2022**, *12*, 1103. https://doi.org/10.3390/app12031103

Academic Editor: Christos Bouras

Received: 30 December 2021
Accepted: 19 January 2022
Published: 21 January 2022

1. Introduction

Software-defined networking (SDN) has arisen as a revolutionary networking paradigm that can meet escalating demands of future networking [1]. Unlike a traditional network, it separates the network's control plane from the embedded nodes and replaces the classical control plane based on system embedding with an open and programmable soft control plane [2,3]. Thanks to the openness and programmability of SDN, it has been applied to various fields, such as enterprise networks and data centers. However, the idea of separation of logical control and forwarding function expands the attack surface [4], and the control plane, data plane, and application plane will face security challenges. For example, since the controller has global network visibility, the entire network can be controlled once it is hijacked. With the deep research and broad application of SDN architecture, security has gradually become a fundamental factor restricting its growth [5,6].

Both academia and the industry have made many efforts and proposed many solutions to enhance SDN security. TopoGuard [7] and the correlation-based topology anomaly detection (CTAD) [8] solve topological attacks in SDN. The saturation attacks detector (SA-Detector) [9], vSwitchGuard [10], FTGuard [11], and SDNGuardian [12] are responsible for defending against data plane saturation attacks and flow table overflow attacks. BWManager [13], DETPro [14], FloodDefender [15], and DoSDefender [16] adopt different ideas to realize the detection and defense of distributed denial of service (DDoS)/DoS attacks in SDN. PacketChecker [17] and INSPECTOR [18] focus on solving the problem of packet injection attacks. These existing works provide reliable mechanisms from different aspects

and strengthen SDN security. However, novel attack methods and defense strategies still need to be discovered and studied.

The authors of [17] first proposed the packet injection attack. They point out that attackers can inject fake nodes into the SDN by maliciously manipulating packets. We continue to carry out research on this basis and consider whether attackers can further exploit these non-existing hosts. The answer is positive. On the one hand, the SDN controller with the OpenFlow protocol as the southbound interface maintains the Host Profile file to track the host's location [7]. When attackers inject spoofed packets by modifying the header's information (e.g., media access control, MAC address), it assumes that a new host joins the network and creates a new Host Profile file. On the other hand, the attacker can quickly establish communication with these non-existent hosts due to a lack of verification of *Packet-In* messages. As these false hosts have been added to the SDN network topology, the controller will generally route and forward the associated communication behaviors. Once enough false hosts are injected, an attacker can use them to generate plenty of new flows in a short period. For the control plane, since the SDN switch sends all packets with unknown flows to the controller, it will overload the controller's processing capabilities with high CPU consumption and not respond quickly to the legitimate user. For the data plane, because the controller must install many flow entries for each spoofed flow, the flow table of the switches with scarce resources can easily overflow [16]. In addition, these unmatched flows would consume an excessive bandwidth of the secure channel [19]. These limited resources will become the bottleneck of the network.

To defend against such an attack that injects and utilizes false hosts to perform malicious behavior, we propose the packet injection exploiting attack defender (PIEDefender). As a lightweight component of the SDN controller, PIEDefender comprises three modules: Injection detection, simplified DoS detection, and flow rule management. The injection detection module constructs and maintains the mapping information between the data plane switch and the host. It identifies malicious hosts who inject fake hosts by verifying information consistency. The simplified DoS detection module detects false nodes based on OpenFlow message and flow features. The flow rule management module is responsible for installing blocking rules against the attack source. These modules cooperate to protect the SDN network effectively.

The contributions of this paper can be summarized as follows:

- We propose the packet injection exploiting attack against SDN. To prove the feasibility of this attack, we carry out experiments in a software environment. The results demonstrate that this attack can effectively affect the control plane and the data plane.
- In order to defend against the packet injection exploiting attack, we design and achieve PIEDefender on the SDN controller. This scheme is protocol-independent and does not require additional hardware equipment and any data plane modification.
- We evaluate the PIEDefender in the injection detection precision, DoS detection precision, and defense overhead. The results demonstrate that the PIEDefender effectively mitigates the packet injection exploiting attack with limited overhead.

The rest of the paper is organized as follows. In Section 2, we survey the background and related work. We demonstrate the feasibility of the packet injection exploiting attack and analyze the consequences in Section 3. We detail the implementation of the PIEDefender in Section 4 and performance evaluation in Section 5. Finally, we conclude our work in Section 6.

2. Background and Related Works

2.1. Background

SDN separates the control layer from the data layer of the network and replaces the original routing negotiation method with a centralized controller, which significantly improves the efficiency and flexibility of network management and control. Figure 1 shows a typical SDN network architecture consisting of the data plane, control plane, application plane, southbound, and northbound interfaces. The data plane is running for forward-

ing data packets according to specified rules or policies. The control plane maintains the resources, global topology visibility, and status information. The application plane interacts with the controller through the northbound application rpogramming interface (API), facilitating the rapid advancement of services, such as network configuration and application deployment.

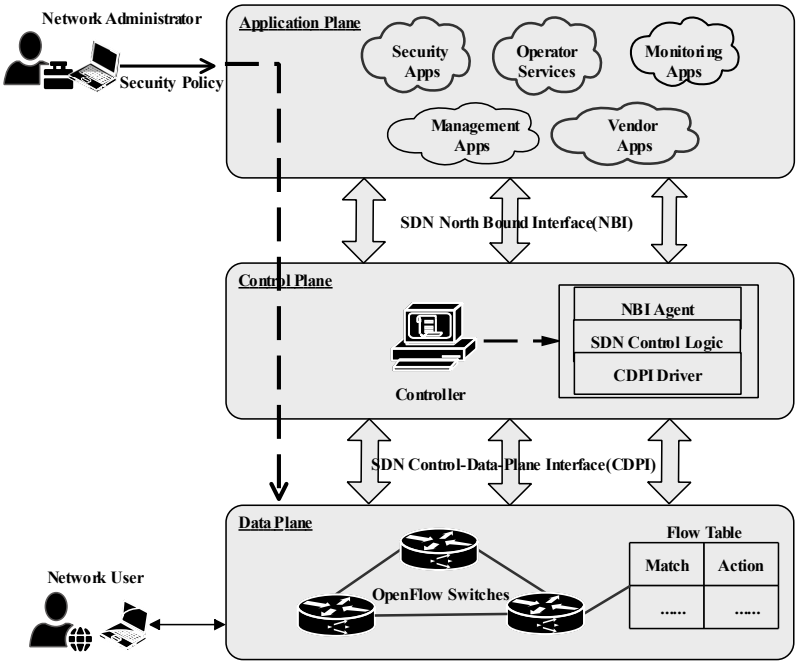


Figure 1. Software-defined networking (SDN) architecture [20].

Due to the centralized control, the topology management service in SDN is different from a traditional network. To promote network management, the OpenFlow controller maintains topology information and provides this visibility for upper-layer services/applications [7]. In SDN, topology management mainly includes switch discovery, host discovery, and link discovery. For host discovery, the controller tracks host location by maintaining the Host Profile. It listens to *Packet-In* messages and indexes the Host Profile to handle host mobility.

2.2. Related Works

Due to centralized control and programmability features, SDN can provide more advanced functions, such as network monitoring, flow control, and security analysis. Therefore, early research mainly focused on using SDN to empower traditional networks. With the widespread application of SDN, its security problems have become increasingly prominent, and an increasing number of scholars have researched the security of SDN from different aspects.

For the topology security in SDN, TopoGuard [7] can automatically discover topology poisoning attacks in real-time. However, it ignored the origin of *Packet-In* messages in handling the host location hijacking attack. In [21], Hauth proposed solving the host usurp attack. It implemented the confirmation for legal hosts by providing an authentication server and authentication log. However, Hauth requires additional measures to ensure the authentication server does not become damaged. In [22], the authors demonstrated the persona hijacking, and presented SECUREBINDER, which prevented identifier binding problems. What needs to be emphasized is that the target must be using DHCP for Persona Hijacking to be applicable. CTAD [8] detected different topology attack types by analyzing the relevance of network traffic and verifying link layer discovery protocol (LLDP) frames. TrustTopo [23] as a lightweight and efficient SDN topology verification scheme, coped with

the host hijacking and link fabrication attacks. However, this schema did not consider the threats in the startup stage of the network.

For resource saturation attacks in SDN, the SA-Detector [9] calculated the self-similarity of regular and abnormal traffic and analyzed the difference against saturation attacks. However, this scheme did not considered the origin of the attack during a saturation attack. In [12], the authors described an enhanced saturation attack and proposed an efficient defense framework named SDNGuardian. vSwitchGuard [10] aimed to identify the compromised switches targeted by foregone or unknown types of saturation attacks with machine learning classifiers. However, the paper only studied five saturation attacks in the SDN environment, and more types of attacks need to be investigated.

For the flow table security in SDN, FTGuard [11] implemented a behavior-based priority-aware defense strategy to cope with the flow table overflow attack. WedgeTail [24] distinguished malicious forwarding devices by computing the expected and actual trajectories of packets, effectively protecting the data plane. This approach is helpful, but its deployment in a real-world network is a challenge. The authors of [25] proposed a flow table sharing mechanism, which effectively mitigates the damage to the normal network caused by the flow table overloading attack. In [26], the authors proposed a quality of service (QoS) aware mitigation mechanism, which combined all available flow table resource to solve overloading attacks on a single switch of the system. However, due to the timeout operation, legitimate flows faced the possibility of being denied into the network.

For research on denial of service (DoS) attacks in SDN, the authors of [27] proposed a machine learning approach to detect a DoS attack on SDN data plane switches using flow-table information and OpenFlow traffic. In [28], the authors build a mechanism that monitors the network and can differentiate DoS traffic from benign traffic using entropy in an SDN environment. Neural networks were used to detect DoS attacks in [29,30]. SDN-Guard [31] and FloodDefender [15] mitigated DoS attacks in SDN by dynamically managing flow rules. Although they performed well in defending against DoS attacks, they required additional hardware equipment. In addition, the BWManager [13] proposed an innovative scheduling algorithm for a controller based on bandwidth prediction. As an extension component of the SDN controller, DoSDedender [16] defended against DoS attacks by maintaining the mapping relationship between switches and associated hosts. This solution uses a threshold-based approach to detect attacks that falsify source ports, easily leading to false positives.

This paper focuses on detecting and mitigating the packet injection exploiting attack. We will introduce it in detail in later sections.

3. Packet Injection Exploiting Attack

This section describes the packet injection exploiting attack in the SDN environment. We demonstrate how attackers can use injected hosts to launch DoS attacks.

3.1. Threat Model

We suppose the attacker controls at least one compromised physical or virtual host. This hypothesis is reasonable because it is compatible with the previous work [13,17]. Besides, we also assume that the SDN controller runs in reactive mode, widely adopted by most common controllers, such as Floodlight, Ryu, and Pox. In reactive mode, the controller communicates with the switch over the secure channel. For example, the controller sends instructions and installs forwarding rules to the OpenFlow switch. The switches encapsulate and forward unknown packets to the controller. In such a case, attackers can forge lots of unknown packets and continuously trigger the matching action without needing more privileges than legitimate users [16].

We describe a simple attack scenario shown in Figure 2, including an SDN controller, an OpenFlow switch, and two compromised hosts. The OpenFlow switch forwards unknown packets to the controller through *Packet-In* messages, and the controller installs flow rules through *Flow-Mod* and *Packet-Out* messages. The attacker injects fake hosts by host *A*

to generate a ghost topology and carries out DoS attacks through host *B*. The core attack process includes:

(1). The attacker sends plenty of packets with forged source MAC addresses to the SDN network through host *A*.

(2). When the switch receives an unknown data packet from host *A*, there is no corresponding forwarding rule in the flow table. It will encapsulate the packet into a *Packet-In* message and forward it to the controller.

(3). When receiving a *Packet-In* message from the switch, the SDN controller will retrieve the Host Profile file. Since the attacker has spoofed the source MAC address, as described in Section 2, the host tracking service will think a new host has joined the network, thus adding a fake host to the SDN topology.

(4). After injecting many fake hosts, the attacker can further construct specific packets through host *B*, specify the destination MAC address as a forged fake MAC address, randomly generate the destination IP or port, and send it to the SDN network.

(5). Since the fake hosts have been added to the SDN topology, the controller will not delete these unknown packets from host *B*. This is the most significant difference from the attack by faking the destination MAC address. The controller calculates the forwarding strategy of unknown packets and installs flow table rules to the switch.

(6). If the attacker frequently sends unknown packets through host *B*, it will overload the controller's processing capabilities and the flow table of switches. In addition, these unmatched flows would consume excessive bandwidth of the southbound interface, and even affect the entire network.

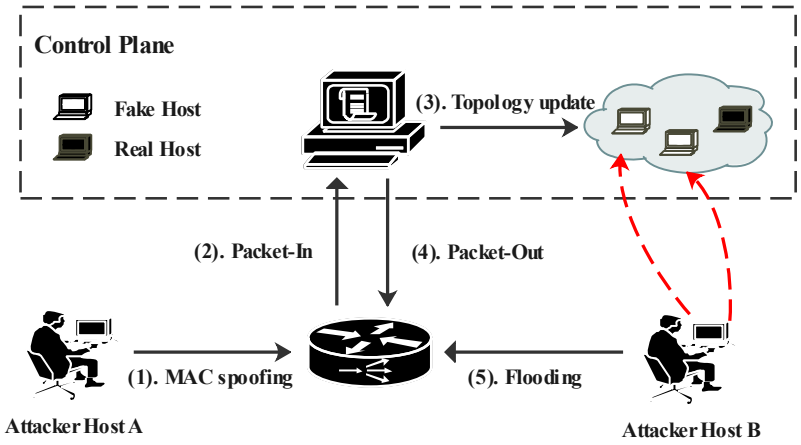


Figure 2. Attack scenario.

3.2. Threat Experiment

We conduct experiments in a software environment to verify the feasibility and effectiveness of the packet injection exploiting attack. The experiment topology is simulated by Mininet, as shown in Figure 3. We select Floodlight v1.2 as the controller, and the southbound interface is Openflow1.3. We select *h1* and *h6* as attacking hosts and others as normal network users to simulate the attack. The host *h1* uses Scapy [32], a powerful interactive packet manipulation program to inject fake hosts into the SDN network, and the host *h2* implements a DoS attack based on these fake hosts.

Next, we carry out verification from the injection of fake hosts and the implementation of DoS attacks.

(1) Packet Injection Attack: The host *h1* forges data packets and injects them into the experimental network at a fixed rate. The source MAC address of the forged data packet is generated using the built-in *RandMAC()* function of Scapy, and the destination MAC and IP address point to the host h6. We modify the source code to output the device MAC since Floodlight does not display additional device information intuitively. The experimental results are shown in Figure 4. As we can see, the injected host is successfully learned by the

device management service and added to the SDN network topology. In this way, we inject 50 non-existent devices.

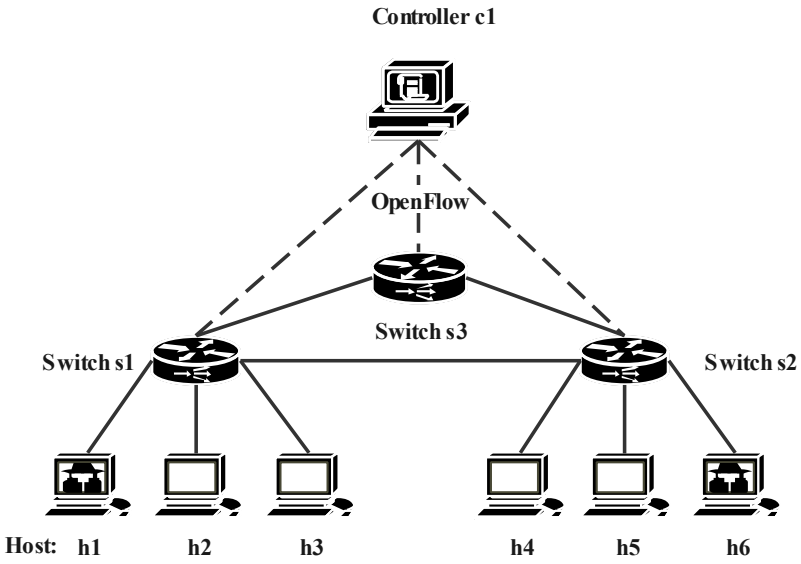


Figure 3. Environment setup.

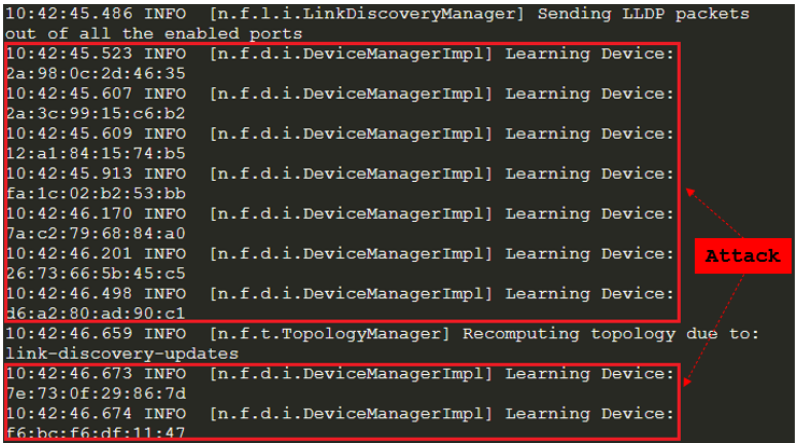


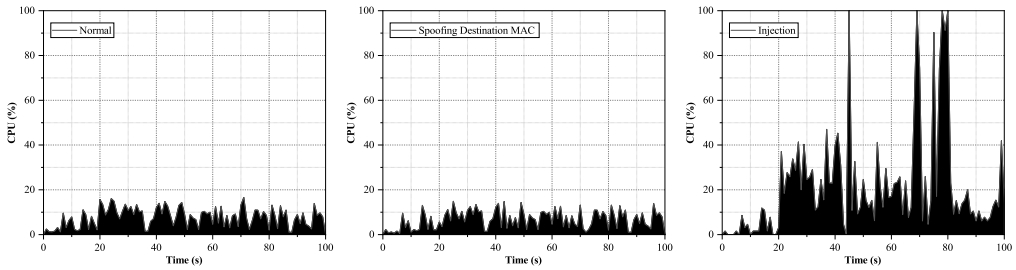
Figure 4. Floodlight with injection attack.

(2) DoS Attack: Since the attacker knows the fake hosts, they can use them to implement a denial-of-service attack by constructing specific packets on the host *h6*. These packets have two characteristics: (1) The destination MAC points to the MAC address of any injected host and (2) the randomly generated IP address and port number. It should be emphasized that there are two main reasons why we call it DoS instead of DDoS. On the one hand, these injected hosts are non-existent. Their topological location is related to the host that launched the injection attack, and they share the same bandwidth resources. On the other hand, neither the attack's power, scale, or destructiveness is the same as DDoS.

When the switch receives the attack packets from the host *h6*, there is no corresponding processing rule in the flow table, and it will forward them to the controller by *Packet-In* messages. Since the injected fake host has been added to the SDN network topology, the controller will calculate and install the forwarding strategy without discarding the packets associated with them. This is the main difference with the forging destination MAC address attack. We evaluated the attack's impact from the two aspects of controller CPU utilization and switch flow table occupancy. At the same time, we compared the forging destination MAC address attack to illustrate the difference between the two.

Attack impact on the SDN controller: We measured the controller CPU usage using the Psutil library [33] under different conditions. Figure 5a–c respectively represent the

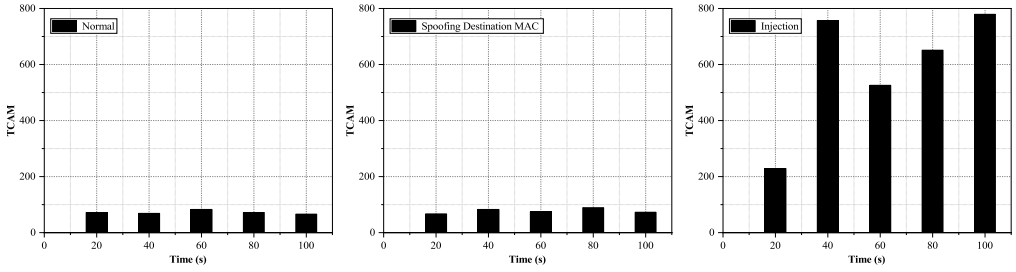
impact on the controller CPU under the normal network status, the forging destination MAC address attack, and the packet injection exploiting attack. As shown in Figure 5a,b, the controller CPU consumption under the forging destination MAC address attack is the same as the normal network state. This is because the manipulated MAC address does not exist in the SDN network topology, and the controller discards the packets without making a decision. As we can see from Figure 5c, the CPU utilization rate rises significantly under the packet injection exploiting attack. The main reason is that all manipulated hosts have been injected into the SDN topology, and the controller will compute routes for unmatched packets about them. Thus the controller load increases sharply due to frequent decision-making.



(**a**) (**b**) (**c**)

Figure 5. Attack impact on controller CPU usage. (**a**) Normal. (**b**) Spoofing Destination MAC. (**c**) Injection.

Attack impact on OpenFlow switch: We counted the occupancy of the switch flow table under different conditions, and the results are shown in Figure 6. As mentioned above, when an attacker implements the forging destination MAC address attack, the controller will delete unmatched packets. Since no more flow entries are installed, the flow table occupancy under the forging destination MAC attack is similar to the normal network state, as shown in Figure 6a,b. Figure 6c shows the occupancy of the flow table under the packet injection exploiting attack. As we can see, the number of flow table entries increases rapidly, reaching a maximum of 800. This is because the controller calculates the forwarding path for unmatched packets and installs plenty of flow rules.



(**a**) (**b**) (**c**)

Figure 6. Attack impact on OpenFlow switch. (**a**) Normal. (**b**) Spoofing Destination MAC. (**c**) Injection.

4. Countermeasures

4.1. System Architecture

The PIEDefender stands between the controller platform and other controller apps, as depicted in Figure 7. It includes three function modules: Injection detection, simplified DoS detection, and flow rule management. The injection detection module builds and maintains the mapping relationship between the switches and the hosts, and it identifies malicious hosts with injection behaviors by verifying the consistency of information. The simplified DoS detection module utilizes OpenFlow message and flows features to

detect hosts that use injected hosts to carry out a DoS attack. Once a malicious node is detected, the injection detection module and simplified DoS detection module will give the threat information to the FlowRule Management module, which installs blocking rules against the attack source, effectively protecting the SDN network.

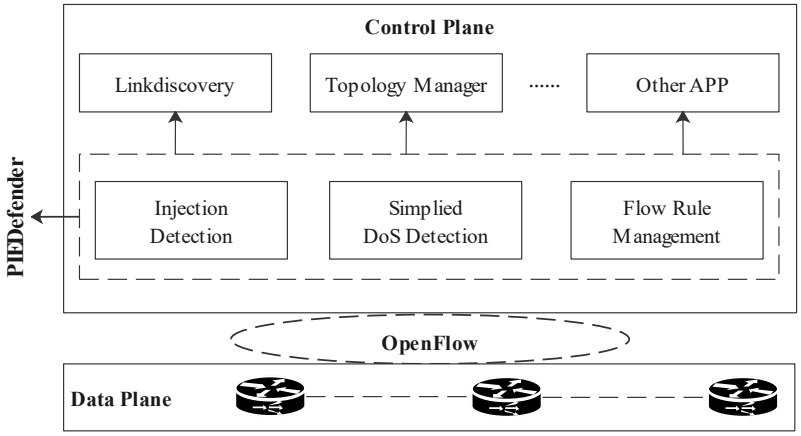


Figure 7. Architecture of the packet injection exploiting attack defender (PIEDefender).

For better understanding, we use Table 1 to show the meanings of major notations. Next, we will introduce the functions of each module in detail.

Table 1. Notations.

Notations	Definition
S	Switches in the network
s_j	Switch j
h_i	Host i
mac_i	MAC address of h_i
$dpid_i$	DPID of of s_j
$port_{ij}$	Port which connected from h_i to s_j
MT	Mapping table of switches and hosts
$interval$	Time interval
$PINum$	Number of Packet-In messages
$PMNum$	Number of Packet-Mod messages
$IFNum$	Number of irreversible flows
$FlowNum$	Number of flows
$Flow_i$	The i_{th} flow
T_{PN}	Threshold of packets number
$PacketsNum_i$	Total packets of i_{th} flow
T_{FL}	Threshold of flow lifetime
$Lifetime_i$	Lifetime of i_{th} flow

4.2. Injection Detection

Attackers can easily inject fake hosts by maliciously injecting manipulated packets into SDN. The injection detection module monitors and implements the verification of *Packet-In* messages and *OFPT_PORT_STATUS* messages to detect this attack. By verifying the *Packet-In* message, it is possible to find whether an injection attack occurs. By ascertaining the *OFPT_PORT_STATUS* message, it is possible to track host location changes dynamically and prevent attackers from bypassing the static matching strategy.

(1) *Packet-In* Message Verification: The injection detection module constructs and maintains the mapping table MT between the switch and the host connected. Each mapping entry in MT is composed of mac_i, $dpid_j$, and $port_{ij}$. Among them, mac_i represents the MAC address of host h_i, $dpid_j$ represents the DPID (datapath ID) of switch s_j, and $port_{ij}$ represents the port number where host h_i is connected to switch s_j. The verification process

is shown in Algorithm 1. When receiving a *Packet-In* message from switch s_j, the injection detection module first extracts mac, dpid, and port. Then according to dpid and port, it queries mac_i from *MT*. It needs to be emphasized that the compromised target for the packet injection exploiting attack is the host, not the switch. Since the switch generates the *Packet-In* message, the dpid and port where the host connects to the switch are factual information, and the attacker spoofed only the host's mac address. There are three cases in the verification process.

Case 1: There is no entry found in *MT*.

For this case, we consider a new host to join the network or migrate to a new location. We will create a new entry in *MT* and forward the message to subsequent modules for processing.

Case 2: There is an entry in *MT*, and the mac_i and mac are the same.

For this case, we think that the *Packet-In* message is legal and then forward it to the next module for processing. The reason is that the host's identification has been consistent since joining the network.

Case 3: There is an entry in *MT*, but the mac_i is different from mac.

For this case, we consider that an injection attack occurs and then forwards the malicious host information to the flow Rule management module. We agree that the host information may change. For example, a host location migrates, or the network user changes the host mac address. The latest mapping information will be stored in *MT* if it is a regular migration after passing *port-status* message verification. Therefore, we have strong reasons to believe that an attack occurs when the information is inconsistent.

Algorithm 1 *Packet-In* Message Verification

Input: *S*: set of switches, *MT*: mapping table

1: **For** $\forall s_j \in S$:
2: Listening *Packet-In* message from s_j
3: Extract mac, dpid, port from *Packet-In* message
4: Get mac_i from *MT* by dpid and port
5: **If** mac_i not exist in *MT*:
6: Set mac_i=mac and add to *MT*
7: Forwarding and Continue
8: **Else If** mac_i == mac:
9: Forwarding and Continue
10: **Else**: Report entry to flow rule management
11: **End For**

(2) *Port-Status* Message Verification: Inspired by the DoSDefender [16], the injection detection module monitors the *OFPT_PORT_STATUS* message to realize the dynamic management of the mapping information between the switch and connected hosts. This is because when a host location migrates or the status of a switch port changes, the controller receives an *OFPT_PORT_STATUS* message. The entry containing the corresponding switch and host will be deleted simultaneously based on this message. However, since the *OFPT_PORT_STATUS* message can be actively triggered (such as restarting the network service), only through this message to manage *MT* has the risk of being bypassed. For example, we assume a compromised host h_1, whose MAC address is m_1. The attacker can bypass the defense in two steps.

Step 1: The attacker actively triggers the *OFPT_PORT_STATUS* message, causing the defense mechanism to delete the mapping entries related to h_1.

Step 2: The attacker sends crafted packets with spoofing mac address m_2 to trigger *Packet-In* messages. When receiving a *Packet-In* messages, the defense mechanism will create a new entry related to h_1, whose mac is m_2.

In the whole process, we can see that the mac address of h_1 has not changed, but a fake host with mac address m_2 has been successfully injected into the network.

For the above reason, the injection detection module implements the legality verification, as shown in Algorithm 2. The primary idea is that if a host's mapping is updated, the address information should be consistent with the latest mapping in its subsequent packets. After receiving the *OFPT_PORT_STATUS* message, it creates a temporary mapping table MT' to store the switch and host information in the subsequent related *Packet-In* messages. If MT' contains only one mapping entry when the *Packet-In* messages reach the set threshold, it indicates that the message is legal and updates the MT. Before reaching the threshold, the original mapping entry in the MT will not be deleted. Otherwise, it is considered that the *OFPT_PORT_STATUS* message is illegal and forwards host information to the flow rule management module.

Algorithm 2 *Port-Status* Message Verification

Input: S: set of switches, MT: mapping table,
$counter$: *packet-in* message verify threshold

For $\forall s_j \in S$:
Listening *oftp_port_status* message from s_j
Construct temporary mapping table MT'
For *i* = 1 to *counter*:
Listening *packet-in* message from s_j
Extract mac, dpid, port from *packet-in* message
Encapsulate mapping entry which key is mac and value is dpid and port and insert to MT'
End For
If MT' only contains one mapping entry:
Update MT by the mac, dpid, and port
Else: Report entry to flow rule management
End For

4.3. Simplified DoS Detection

The injection detection module authenticates the *Packet-In* and *OFPT_PORT_STATUS* messages and realizes the detection of injection attacks. However, the detection accuracy is related to the threshold setting, and there is still the possibility of successful injection. Therefore, PIEDefender implements the simplified DoS detection module to detect malicious hosts that conduct DoS attacks based on injected hosts.

The attacker only needs to construct specific packets with MAC addresses pointing to these injected hosts and randomly forge the IP and port, and they can generate numbers of new flows soon. In this way, these unmatched flows would overload the flow table of the switches and excessively consume the controller CPU and the secure channel bandwidth. We extracted the five-tuple as the feature vector to indicate whether the attack occurred by comprehensively analyzing the attack method and the effect. The detailed description and judgment methods are as follows.

(1) Rate of *Packet-In* messages (RPI)

When a DoS attack occurs, the switch will frequently request the controller via *Packet-In* messages, increasing the message rate significantly. Therefore, we take the rate of *Packet-In* messages (RPI) as an essential parameter to identify the occurrence of an attack and use the following formula for calculation:

$$RPI = \frac{PINum}{interval} \tag{1}$$

where $PINum$ denotes the sum number of *Packet-In* messages, and $interval$ indicates the time interval.

(2) Rate of *Flow-Mod* messages (RFM)

When an attacker launches a DoS attack, the controller must install some flow entries via *Flow-Mod* messages to establish routes for spoofed flows, increasing the message rate significantly. Therefore, the rate of *Flow-Mod* messages is an essential characteristic to distinguish attack traffic. The equation of RFM is as follow:

$$RFM = \frac{FMNum}{interval} \tag{2}$$

where $FMNum$ denotes the sum number of *Flow-Mod* messages.

(3) Percentage of irreversible flows ($PIRF$)

We define that the two flows $flow_x$ and $flow_y$ are irreversible if they satisfy any of the following conditions:

- SrcIP($flow_x$) $\neq$ DstIP($flow_y$);
- DstIP($flow_x$) $\neq$ SrcIP($flow_y$);
- SrcPort($flow_x$) $\neq$ DstPort($flow_y$);
- DstPort($flow_x$) $\neq$ SrcPort($flow_y$).

An attacker sends false packets to implement DoS attacks, only to ensure that the MAC addresses of the packets point to injected hosts and randomly generate IP addresses and ports number. In this case, they cannot establish a complete two-way connection with the target. Thus, the percentage of irreversible flows ($PIRF$) sharply increases during the attack and use the following formula for calculation:

$$PIRF = \frac{IFNum}{FlowNum} \tag{3}$$

where $IFNum$ denotes the number of irreversible flows, and $FlowNum$ denotes the sum number of flows.

(4) Percentage of flows with small packets ($PFSP$)

A DoS attack usually generates plenty of new flows with a high rate, and the number of packets in each flow is minor (e.g., about 1∼3 packets per flow) [13]. Therefore, we can select the percentage of flows with small packets to reveal attack severity. The equation of $PFSP$ is as follow:

$$PFSP = \frac{\sum_{i}^{FlowNum} Flow_i(PacketsNum_i < T_{PN})}{FlowNum} \tag{4}$$

where $Flow_i$ is the i_{th} flow, $PacketsNum_i$ denotes the packets number of i_{th} flow, and T_{PN} is a threshold value.

(5) Percentage of flows with a short lifetime ($PFSL$)

Since most attack packets with the same information appear only once, corresponding flow rules installed by the controller will not stay for a long time before timeout. When a DoS attack occurs, massive invalid packets make the percentage of flows with a short lifetime increases. The equation of $PFSL$ is as follows:

$$PFSL = \frac{\sum_{i}^{FlowNum} Flow_i(Lifetime_i < T_{FL})}{FlowNum} \tag{5}$$

where $Lifetime_i$ represents the duration of the flow rule for i_{th} flow and T_{FL} denotes the threshold value of a time duration.

The DoS detection module implements a detection scheme based on machine learning (ML) using the above five-tuple feature. At present, ML has become an effective technology

to provide system security and is widely used [14,15,27,34]. Unlike traditional solutions, ML enables the network to identify attacks automatically. It can not only detect known attacks but also identify unknown threats. At the same time, considering the number of features, complexity, etc., we use support vector machine (SVM) as the classifier. SVM is a robust supervised learning method used for classification, regression, and outliers detection, being little affected by noisy data.

The simplified attack detection mechanism is shown in Algorithm 3. When the classification result indicates that an attack occurs, it will notify the injection detection and flow rule management modules. The injection detection will clear and rebuild the MT, and the flow rule management module installs blocking rules to prevent malicious traffic from entering the SDN network.

Algorithm 3 Attack Detection

Input: S: set of switches, H: set of hosts,
ΔTI: time interval,
ΔPN: packets number threshold of per flow,
ΔFL: lifetime threshold of per flow
Output: SV: set of vulnerable hosts

1: Initialize $SV = \varnothing$ and store vulnerable hosts
2: **For each switch** s_j **in** S:
3: Collect dataset ds in T_{TI} time window based on s_j
4: **End For**
5: **For each host** h_i **in** H:
6: Extract dataset ds_i about h_i from ds
7: Compute $feature_i$ based on ds_i, ΔTI, ΔPN, ΔFL
8: Detect attack with Classifier and $feature_i$
9: **If** detection result is **ATTACK**:
10: Add the vulnerable host h_i to SV
11: **End For**

4.4. Flow Rule Management

The flow rule management module receives notice from the injection detection module and the simplified DoS detection module. It extracts the compromised host and installs flow rules to the switch connected to block and removes malicious traffic. As described in OpenFlow specification 1.3, any packet that matches the flow rule will be dropped [35] if no action is specified in a flow entry. Thus, the flow rule management module can construct a *Flow-Mod* message with no output action and can send it to the specified switch to remove the flows.

5. Evaluation

This section implements PIEDefender and evaluates its performance and defense overhead in software environments.

5.1. Environment

Experimental Setup: The experimental topology is shown in Figure 3, simulated by Mininet, using the OpenFlow1.3 protocol as the southbound interface, running on an Ubuntu virtual machine with an Intel Core i5-8400 2.80 GHz CPU and 8 GB of memory. We select *h1* and *h6* as attackers. The *h1* is responsible for launching the packet injection attack, and *h6* implements DoS attacks based on injected hosts. Based on this, we evaluate the performance and overhead of the PIEDefender.

Parameters Setting: To evaluate the injection detection module, we capture 1 h of normal traffic and count the changes in the number of data packets in each flow. After comprehensively considering the verification efficiency and effect, we set the threshold *counter*

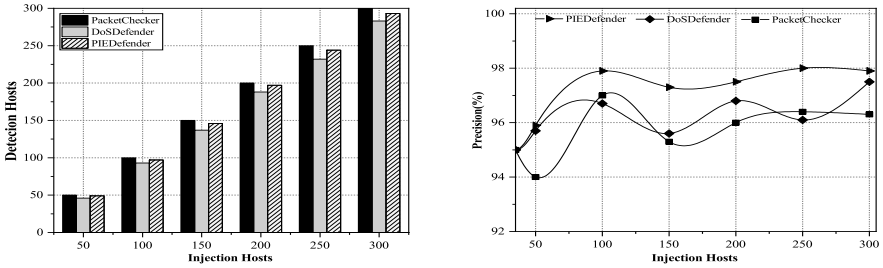
of the *OFPT_PORT_STATUS* message verification process to 8. We can change the value by evaluating the actual network state.

Dataset: To evaluate the simplified DoS detection module, we generate a data set in the environment mentioned above. Through the training sample generation, we collect 30,000 traffic, including 16,925 normal traffic and 13,075 attacks traffic.

5.2. Evaluation

We evaluate the effectiveness and performance of PIEDefender from the following aspects: (1) Injection detection effect, (2) DoS detection effect, (3) defense effect on the SDN controller, (4) defense effect on OpenFlow switch, and (5) defense overhead.

Injection detection effect: We evaluate the injection detection effect of the PIEDefender by comparing the number of injected and detected hosts and compare with PacketChecker [17] and DoSDefender [16]. Figure 8a shows the number of injected hosts successfully discovered. As we can see, PacketChecker can find all injected hosts than PIEDefender because it assumes that the host's MAC address in the data plane will not change. On the one hand, we think this assumption is too strict to some extent. On the other hand, this mechanism cannot distinguish between malicious and non-malicious behavior. The malicious injection has apparent intent to attack, and the purpose is to generate many puppet hosts in the SDN network topology. Otherwise, we consider it as non-malicious behavior. Figure 8b depicts the detection precision and the average detection precision of PacketChecker, DoSDefender, and PIEDefender to be 96.4%, 96.9%, and 97.8%, respectively. It can be seen that the PIEDefender are less affected by the injection scale, and the detection performance is more stable. Although DoSDefender tracks host location migration through *OFPT_PORT_STATUS* messages, the defense mechanism can be easily bypassed due to the lack of message verification. The PIEDefender uses the threshold to verify the legality of *OFPT_PORT_STATUS* messages, preventing attackers from bypassing to a certain extent. Therefore, considering the applicability, stability, and precision, the PIEDefender is superior to other solutions.



(**a**) (**b**)

Figure 8. Comparison of injection detection. (**a**) Detection hosts. (**b**) Detection precision.

DoS detection effect: As mentioned in Section 4.3, we use SVM as the classifier. To prove that SVM is more suitable for our experimental environment and scenarios, we first compared with other classification and clustering algorithms commonly used in anomaly detection, including k-Nearest Neighbors (KNN), Random Forest, Decision Tree, K-means, and BayesNet. As a comparison, we use the F1 score, precision, and recall to evaluate the performance of different algorithms and schemes, as shown in Equations (6)–(8). In Equations (6) and (7):

(1) **TP**: The number of true positives means the classification of the simplified DoS detection module is correct, and an attack occurs.

(2) **FP**: The number of false positives means the classification of the simplified DoS detection module is incorrect and that no attack occurs.

(3) **FN**: The number of false negatives, which means the classification of the simplified DoS detection module is incorrect, and an attack occurs.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1\ score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}. \tag{8}$$

The results is shown in Table 2. As we can see, the SVM algorithm is superior to other detection algorithms in terms of precision, recall, and F1 score for this paper's data and features.

Next, we compare with other proposed solutions in the same environment, including the entropy-based detection method [28], Detpro, and DoSDefender. Detpro uses machine learning methods for detection, and DoSDefender is a static detection solution. We randomly selected the attacker and the timing of the attack, and the evaluated result is shown in Table 3. As we can see, the attack detection accuracy of PIEDefender is superior to the others.

Table 2. Results of different detection algorithms.

Algorithm	Precision	Recall	F1 Score
KNN	94%	94%	94%
SVM	98%	98%	98%
KMeans	96%	96%	96%
BayesNet	95.92%	94%	94.95%
DecisionTree	96.08%	98%	97.03%
RandomForest	90%	90%	90%

Table 3. Results of different detection solutions.

Solution	Precision	Recall	F1 Score
Entropy-based	93.88%	92%	92.93%
Detpro	97.92%	94%	95.92%
DoSDefender	95.92%	94%	94.95%
PIEDefender	97.96%	96%	96.97%

Defense effect on SDN controller: We utilize controller CPU utilization to evaluate the protection on the control plane with the PIEDefender. Figure 9a–d represent the CPU utilization under the injection of a different numbers of hosts. As we can see, the controller CPU usage sharply increases without the PIEDefender when attacks start. It is because the controller needs to handle lots of *Packet-In* requests, and its CPU processing power becomes less available. Since the PIEDefender can effectively detect attacks and install rules to drop malicious traffic without being forwarded to the controller, the controller CPU utilization remained normal before and after the attack under the protection of PIEDefender. At the same time, comparing the experimental results under different injection amounts, PIEDefender has a stable defense ability on controller CPU utilization. In summary, all results show that the PIEDefender can protect SDN controllers effectively.

Defense effect on OpenFlow switch: We measure the number of flow table entries to evaluate the protection on the data plane with the PIEDefender. Figure 10a–d represent the flow table occupation under the injection of different numbers of hosts. As we can see, when the attack starts without the PIEDefender, the number of flow entries of the switch increases rapidly. It is because the controller does not validate the *Packet-In* messages and installs rules for malicious packets. Since PIEDefender can quickly detect attacks and drop malicious traffic, the number of flow entries changes steadily without a surge. At the same time, comparing the experimental results under different injection amounts, PIEDefender has a stable defense ability on the switch flow table. In summary, all results indicate that the PIEDefender can effectively protect the data plane.

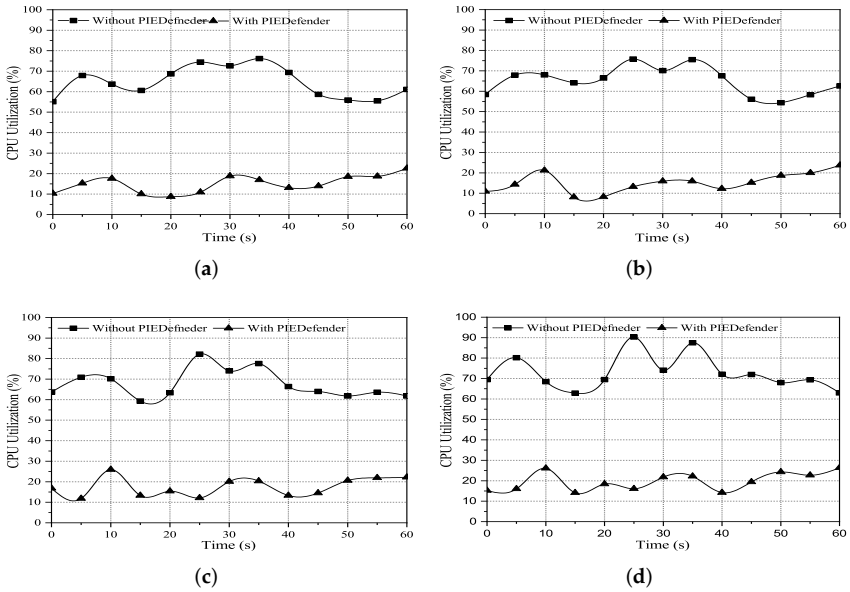


Figure 9. Comparison of controller CPU utilization with and without PIEDefender. (**a**) 50 hosts. (**b**) 100 hosts. (**c**) 150 hosts. (**d**) 200 hosts.

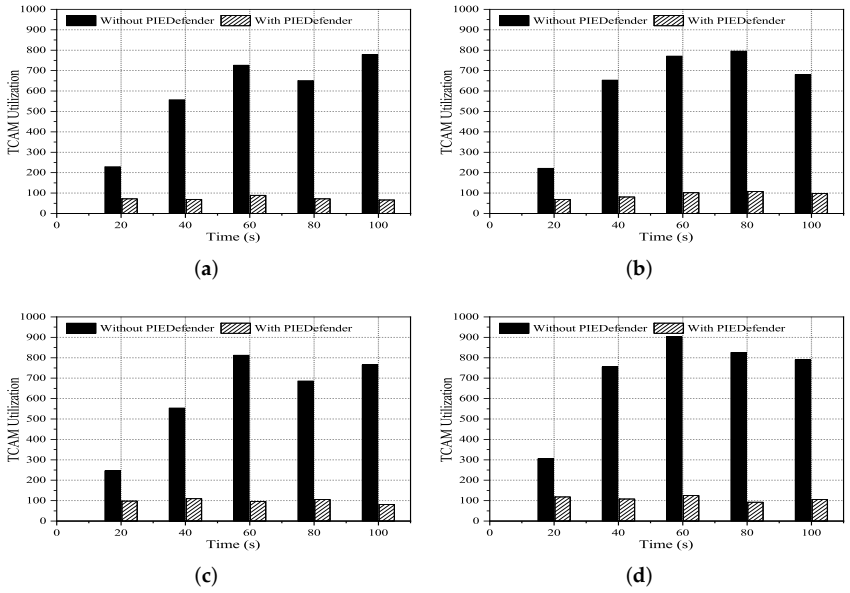


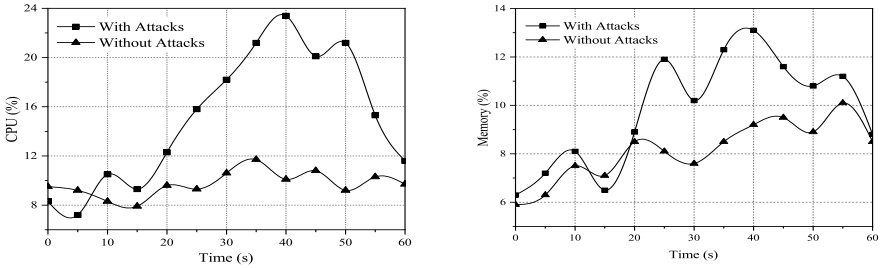
Figure 10. Comparison of switch ternary content addressable memory (TCAM) utilization with and without PIEDefender. (**a**) 50 hosts. (**b**) 100 hosts. (**c**) 150 hosts. (**d**) 200 hosts.

Defence overhead: We measured the CPU and memory resource consumption of PIEDefender with and without attacks to evaluate the defend overhead. The test time was the 60 s, and the attack started in 15 s.

Figure 11a depicts the change in CPU utilization. As we can see, with no attacks, the controller CPU utilization changes smoothly without significant fluctuations. It is mainly related to the implementation of the PIEDefender. On the one hand, the injection detection module uses the hashmap to manage mapping entries, and the complexity of the message verification function is *O(1)*. On the other hand, we have pre-trained the attack detection model and no training process is required after deployment. With attacks, the discovery and detection of malicious hosts require a specific period. As a result, the CPU consumption increases gradually, peaking at 24%. After completing detection, the PIEDefender installs flow rules to discard malicious traffic. Thus, CPU consumption decreases and levels off.

Figure 11b shows the shift in memory utilization. Since the verification function needs to store the mapping relationship between switches and hosts, it will bring some storage overhead. However this is minimal, especially when no attack occurs. There was an approximate 4% increase in memory utilization with attacks. This is because the *Port-Status* message verification needs to store a copy mapping information temporarily, but the impact of this consumption is negligible.

In conclusion, the experimental results demonstrate that the defense overhead of PIEDefender is acceptable.



(**a**) (**b**)

Figure 11. Defence overhead of PIEDefender. (**a**) CPU. (**b**) Memory.

From the above results, we can conclude that the PIEDefender can prevent the packet injection exploiting attack on the control plane and data plane of the SDN network with limited overhead.

Further discussion: We demonstrate that the PIEDefender is effective for mitigating the packet injection exploiting attack. However, it is questionable whether the PIEDefender is useful against other common attacks in SDN. Therefore, we further discuss its applicability. For a certain kind of attack, we think the PIEDefender is applicable if it can, or after extending, defend. Otherwise, it is not applicable. The result is shown in Table 4.

Table 4. Applicability of PIEDefender.

Attack	Related Works	Applicable	Not Applicable
Flow table overflow attack	[11,36–38]	✓	
Flow rule conflict	[39–42]		✓
Packet injection attack	[16,17]	✓	
Fingerprinting attack	[43–47]		✓
Topology poisoning attack	[7,23,48]	✓	
Saturation attack	[10,12,49–51]	✓	
New-flow based DoS attack	[1,16]	✓	
Low-rate DoS attack	[20,52–55]		✓

PIEDefender constructs and maintains the mapping table between switches and hosts connected. Since having the latest global topology status, in addition to detecting malicious injection behaviors, PIEDefender can quickly identify malicious hosts and topology attacks with legitimate network configuration information. That is to say that the PIEDefender is beneficial for preventing topology poisoning attacks and packet injection attacks. Besides, PIEDefender utilizes OpenFlow message and flows features to detect hosts which use injected hosts to carry out a DoS attack. These characteristics are compatible with general high-rate DoS attacks. They will increase the messages rate and generate many flows quickly, like the saturation attack, the flow table overflow attack, and the new flow-based DoS attack. However, the PIEDefender has limitations in defending against other attacks, such as fingerprinting and low-rate DoS attacks. We will focus on these attacks in the future.

6. Conclusions

In this paper, we proposed a new attack that exploits OpenFlow protocol vulnerabilities, named the packet injection exploiting attack. We proved the possibility of the attack through experiments and assessed its impact. In order to protect against the packet injection exploiting attack, we designed PIEDefender, an efficient, protocol-independent component to detect and mitigate attacks effectively. We implementeded the PIEDefender prototype in the Floodlight controller and evaluated the effectiveness in the software environment. The evaluation shows that PIEDefender could effectively mitigate the packet injection exploiting attack against SDN with limited overhead. However, our research is inapplicable to wireless networks. This is because all hosts connect through the access point (AP) in a wireless network, and the AP is a single port. In this case, the proposed injection detection mechanism will fail because we cannot effectively identify every terminal. We will consider more scenarios to make our scheme more universal in future work.

Author Contributions: Conceptualization, J.L.; methodology, J.L.; software, J.L.; validation, J.L., H.Z., and Y.L.; formal analysis, J.L. and T.T.; investigation, J.L. and S.Q.; resources, S.Q., T.T. and H.Z; data curation, J.L. and S.Q.; writing—original draft preparation, J.L.; writing—review and editing, S.Q. and T.T.; visualization, J.L., H.Z. and Y.L.; supervision, H.Z. and Y.L.; project administration, S.Q. and T.T.; funding acquisition, S.Q. and T.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Singh, M.P.; Bhandari, A. New-flow based DDoS attacks in SDN: Taxonomy, rationales, and research challenges. *Comput. Commun.* **2020**, *154*, 509–527. [CrossRef]
2. Shu, Z.; Wan, J.; Li, D.; Lin, J.; Vasilakos, A.V.; Imran, M.A. Security in Software-Defined Networking: Threats and Countermeasures. *Mob. Netw. Appl.* **2016**, *21*, 764–776. [CrossRef]
3. Bera, S.; Misra, S.; Vasilakos, A.V. Software-Defined Networking for Internet of Things: A Survey. *IEEE Internet Things J.* **2017**, *4*, 1994–2008. [CrossRef]
4. Rawat, D.B.; Reddy, S.R. Software defined networking architecture, security and energy efficiency: A survey. *IEEE Commun. Surv. Tutor.* **2016**, *19*, 325–346. [CrossRef]
5. Kalkan, K.; Gur, G.; Alagoz, F. Defense mechanisms against DDoS attacks in SDN environment. *IEEE Commun. Mag.* **2017**, *55*, 175–179. [CrossRef]
6. Yang, H.; Yuan, J.; Li, C.; Zhao, G.; Sun, Z.; Yao, Q.; Bao, B.; Vasilakos, A.V.; Zhang, J. BrainIoT: Brain-Like Productive Services Provisioning with Federated Learning in Industrial IoT. *IEEE Internet Things J.* **2021**. [CrossRef]
7. Hong, S.; Xu, L.; Wang, H.; Gu, G. Poisoning network visibility in software-defined networks: New attacks and countermeasures. *Ndss* **2015**, *15*, 8–11.
8. Chou, L.D.; Liu, C.C.; Lai, M.S.; Chiu, K.C.; Tu, H.H.; Su, S.; Lai, C.L.; Yen, C.K.; Tsai, W.H. Behavior anomaly detection in SDN control plane: A case study of topology discovery attacks. In Proceedings of the 2019 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, 16–18 October 2019; pp. 357–362.
9. Li, Z.; Xing, W.; Khamaiseh, S.; Xu, D. Detecting saturation attacks based on self-similarity of openflow traffic. *IEEE Trans. Netw. Serv. Manag.* **2019**, *17*, 607–621. [CrossRef]
10. Khamaiseh, S.; Serra, E.; Xu, D. vswitchguard: Defending openflow switches against saturation attacks. In Proceedings of the 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 13–17 July 2020; pp. 851–860.
11. Zhang, M.; Bi, J.; Bai, J.; Dong, Z.; Li, Y.; Li, Z. Ftguard: A priority-aware strategy against the flow table overflow attack in sdn. In Proceedings of the SIGCOMM Posters and Demos, Los Angeles, CA, USA, 21–25 August 2017; pp. 141–143.
12. Xu, J.; Wang, L.; Xu, Z. An enhanced saturation attack and its mitigation mechanism in software-defined networking. *Comput. Netw.* **2020**, *169*, 107092. [CrossRef]
13. Wang, T.; Guo, Z.; Chen, H.; Liu, W. BWManager: Mitigating denial of service attacks in software-defined networks through bandwidth prediction. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1235–1248. [CrossRef]

14. Chen, Y.; Pei, J.; Li, D. Detpro: A high-efficiency and low-latency system against ddos attacks in sdn based on decision tree. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 22–24 May 2019; pp. 1–6.
15. Shang, G.; Zhe, P.; Bin, X.; Aiqun, H.; Kui, R. FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks. In Proceedings of the IEEE INFOCOM 2017—IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
16. Deng, S.; Gao, X.; Lu, Z.; Li, Z.; Gao, X. DoS vulnerabilities and mitigation strategies in software-defined networks. *J. Netw. Comput. Appl.* **2019**, *125*, 209–219. [CrossRef]
17. Deng, S.; Gao, X.; Lu, Z.; Gao, X. Packet injection attack and its defense in software-defined networks. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 695–705. [CrossRef]
18. Alshra'A, A.S.; Seitz, J. Using inspector device to stop packet injection attack in SDN. *IEEE Commun. Lett.* **2019**, *23*, 1174–1177. [CrossRef]
19. Ni, J.; Zhang, K.; Vasilakos, A.V. Security and Privacy for Mobile Edge Caching: Challenges and Solutions. *IEEE Wirel. Commun.* **2021**, *28*, 77–83. [CrossRef]
20. Tang, D.; Zhang, S.; Yan, Y.; Chen, J.; Qin, Z. Real-time Detection and Mitigation of LDoS Attacks in the SDN Using the HGB-FP Algorithm. *IEEE Trans. Serv. Comput.* **2021**. [CrossRef]
21. Xin, W.; Neng, G.; Zhang, L.C. Hauth: A Novel Approach for Network Visibility Protection. In Proceedings of the International Conference on Computer Networks and Communication Technology (CNCT 2016), Xiamen, China, 16–18 December 2016; Atlantis Press: Amsterdam, The Netherlands, 2016; pp. 128–136.
22. Jero, S.; Koch, W.; Skowyra, R.; Okhravi, H.; Nita-Rotaru, C.; Bigelow, D. Identifier binding attacks and defenses in software-defined networks. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 415–432.
23. Huang, X.; Shi, P.; Liu, Y.; Xu, F. Towards trusted and efficient SDN topology discovery: A lightweight topology verification scheme. *Comput. Netw.* **2020**, *170*, 107119. [CrossRef]
24. Shaghaghi, A.; Kaafar, M.A.; Jha, S. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; pp. 849–861.
25. Siyi, Q.; Chengchen, H.; Hao, L.; others. A mechanism of taming the flow table overflow in OpenFlow switch. *Chin. J. Comput.* **2018**, *41*, 2003–2015.
26. Yuan, B.; Zou, D.; Yu, S.; Jin, H.; Qiang, W.; Shen, J. Defending against flow table overloading attack in software-defined networks. *IEEE Trans. Serv. Comput.* **2016**, *12*, 231–246. [CrossRef]
27. Abhiroop, T.; Babu, S.; Manoj, B. A machine learning approach for detecting DoS attacks in SDN switches. In Proceedings of the 2018 Twenty Fourth National Conference on Communications (NCC), Hyderabad, India, 25–28 February 2018; pp. 1–6.
28. Carvalho, R.N.; Bordim, J.L.; Alchieri, E.A.P. Entropy-based DoS attack identification in SDN. In Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, 20–24 May 2019; pp. 627–634.
29. De Assis, M.V.; Novaes, M.P.; Zerbini, C.B.; Carvalho, L.F.; Abrãao, T.; Proença, M.L. Fast defense system against attacks in software defined networks. *IEEE Access* **2018**, *6*, 69620–69639. [CrossRef]
30. Arivudainambi, D.; KA, V.K.; Chakkaravarthy, S.S. LION IDS: A meta-heuristics approach to detect DDoS attacks against Software-Defined Networks. *Neural Comput. Appl.* **2019**, *31*, 1491–1501. [CrossRef]
31. Dridi, L.; Zhani, M.F. SDN-guard: DoS attacks mitigation in SDN networks. In Proceedings of the 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), Pisa, Italy, 3–5 October 2016; pp. 212–217.
32. Scapy Projec. Available online: https://scapy.net/ (accessed on 20 October 2021).
33. Psutil. Available online: https://pypi.org/project/psutil/ (accessed on 20 October 2021).
34. Dibaei, M.; Zheng, X.; Xia, Y.; Xu, X.; Jolfaei, A.; Bashir, A.K.; Tariq, U.; Yu, D.; Vasilakos, A.V. Investigating the Prospect of Leveraging Blockchain and Machine Learning to Secure Vehicular Networks: A Survey. *IEEE Trans. Intell. Transp. Syst.* **2021**, 1–18. [CrossRef]
35. Specifications. Available online: https://opennetworking.org/software-defined-standards/specifications/ (accessed on 20 October 2021).
36. Noh, S.; Kang, M.J.; Park, M. Protection against Flow Table Overflow Attack in Software Defined Networks. In Proceedings of the 2021 International Conference on Information Networking (ICOIN), Jeju Island, Korea, 13–16 January 2021; pp. 486–490.
37. Soylu, M.; Guillen, L.; Izumi, S.; Abe, T.; Suganuma, T. NFV-GUARD: Mitigating Flow Table-Overflow Attacks in SDN Using NFV. In Proceedings of the 2021 IEEE 7th International Conference on Network Softwarization (NetSoft), Tokyo, Japan, 27 June–1 July 2021; pp. 263–267.
38. Zhao, X.; Wang, Q.; Wu, Z.; Guo, R. Method for Overflow Attack Defense of SDN Network Flow Table Based on Stochastic Differential Equation. *Wirel. Pers. Commun.* **2021**, *117*, 3431–3447. [CrossRef]
39. Cui, J.; Zhou, S.; Zhong, H.; Xu, Y.; Sha, K. Transaction-Based Flow Rule Conflict Detection and Resolution in SDN. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–9.

40. Zhou, Q.; Yu, J.; Li, D. TSSBV: A Conflict-Free Flow Rule Management Algorithm in SDN Switches. In Proceedings of the 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), Helsinki, Finland, 25–28 April 2021; pp. 1–5.
41. We, L. Rule Conflict Detection in Protocol-Oblivious Forwarding. *Microelectron. Comput.* **2015**, *32*, 78–81.
42. Hao, W.; Jiang, Y.; Gao, J. Detection mechanisms of rule conflicts in SDN based on a path-tree model. In Proceedings of the 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 24–26 November 2017; pp. 336–339.
43. Ahmed, B.; Ahmed, N.; Malik, A.W.; Jafri, M.; Hafeez, T. Fingerprinting SDN Policy Parameters: An Empirical Study. *IEEE Access* **2020**, *8*, 142379–142392. [CrossRef]
44. Azzouni, A.; Braham, O.; Nguyen, T.M.T.; Pujolle, G.; Boutaba, R. Fingerprinting OpenFlow Controllers: The First Step to Attack an SDN Control Plane. In Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016; pp. 1–6.
45. Zeitlin, Z.J. Fingerprinting Software Defined Networks and Controllers. 2015. Available online: https://xs2.dailyheadlines.cc/scholar?q=Fingerprinting+Software+Defined+Networks+and+Controllers (accessed on 30 November 2021).
46. Zhang, M.; Hou, J.; Zhang, Z.; Shi, W.; Qin, B.; Liang, B. Fine-Grained Fingerprinting Threats to Software-Defined Networks. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, 1–4 August 2017; pp. 128–135.
47. Cao, J.; Yang, Z.; Sun, K.; Li, Q.; Xu, M.; Han, P. Fingerprinting SDN Applications via Encrypted Control Traffic. In Proceedings of the RAID, Beijing, China, 23–25 September 2019.
48. Chou, L.D.; Liu, C.C.; Lai, M.S.; Chiu, K.C.; Tu, H.H.; Su, S.; Lai, C.L.; Yen, C.K.; Tsai, W.H. Behavior Anomaly Detection in SDN Control Plane: A Case Study of Topology Discovery Attacks. *Wirel. Commun. Mob. Comput.* **2020**, *2020*, 8898949. [CrossRef]
49. Li, Z.; Xing, W.; Xu, D. Detecting Saturation Attacks in Software-Defined Networks. In Proceedings of the 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), Miami, FL, USA, 9–11 November 2018; pp. 163–168.
50. Khamaiseh, S.; Alsmadi, I.; Al-Alaj, A. Deceiving Machine Learning-Based Saturation Attack Detection Systems in SDN. In Proceedings of the 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Leganes, Spain, 10–12 November 2020; pp. 44–50.
51. Huang, X.; Xue, K.; Xing, Y.; Hu, D.; Li, R.; Sun, Q. FSDM: Fast Recovery Saturation Attack Detection and Mitigation Framework in SDN. In Proceedings of the 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Delhi, India, 10–13 December 2020; pp. 329–337.
52. Cao, J.; Xu, M.; Li, Q.; Sun, K.; Yang, Y.; Zheng, J. Disrupting SDN via the Data Plane: A Low-Rate Flow Table Overflow Attack. In *SecureComm 2017: Security and Privacy in Communication Networks*; Springer: Berlin/Heidelberg, Germany, 2017.
53. Pascoal, T.A.; Dantas, Y.G.; da Fonseca, I.E.; Nigam, V. Slow TCAM Exhaustion DDoS Attack. In *SEC 2017: ICT Systems Security and Privacy Protection*; Springer: Berlin/Heidelberg, Germany, 2017.
54. Xie, S.; Xing, C.; Zhang, G.; Zhao, J. Research on Table Overflow Ldos Attack Detection and Defense Method in Software Defined Networks. In *ICBDS 2019: Big Data and Security*; Springer: Berlin/Heidelberg, Germany, 2019.
55. Wu, Z.; Xu, Q.; Wang , J.; Yue, M.; Liu, L. Low-Rate DDoS Attack Detection Based on Factorization Machine in Software Defined Network. *IEEE Access* **2020**, *8*, 17404–17418. [CrossRef]