Watermarking for Detecting Freeloader Misbehavior in Software-Defined Networks

Younghee Park*
Computer Engineering
San Jose State University
younghee.park@sjsu.edu

Sang-Yoon Chang*
Advanced Digital Sciences Center
Singapore
sych@adsc.com.sg

Lavanya M. Krishnamurthy
Computer Engineering
San Jose State University
lavanya.krishnamurthy@sjsu.edu

Abstract—Software-defined networking (SDN) provides network operators a high level of flexibility and programmability through the separation of the control plane from the data plane. When initiating traffic, users are required to install flow rules that direct the traffic in forwarding. This process requires communication between control and data plane and results in significant overhead and enables the controller to monitor the traffic and its source. In this paper, we introduce a novel misbehavior, called freeloading, where attackers bypass the process of installing flow rules. The attackers thus can send traffic with an unfair advantage in delay (enabling them to launch more timely threats) and significantly reduce the risk of attacker detection by the network controller (especially if further threats were launched). To prevent such attack, we develop a flow watermarking technique that embeds a secret message into the data payload. It facilitates ownership of the established flow rules, so that only the legitimate owners of flow rules can send packets using their own rules and the network can help detect the misuse cases of the installed flow rules.

Keywords—Software-defined networking, OpenFlow, Open Vswitch, Network-based attacks.

I. INTRODUCTION

Software Defined Networking (SDN) provides highly flexible programming capability and the abstraction of the lower level network functionalities by decoupling the network controller from the data plane [11], [14], [28]. SDN provides an easy operational platform by having a centralized controller to manage the network; the controller has a global view of current network states [28], and can quickly adapt to complex and dynamic networks by reacting to network states and events [39], [40], [41]. However, the shift in networking paradigm due to SDN introduced new sets of security issues, such as DoS attacks on the controller and the data plane [11], [13], [17], [26], poisoning attacks on network visibility [34], and others [15], [16], [30], [36]. Whereas much prior work focused on vulnerabilities and threats on the control plane, there has been limited treatment on the misuse of the data plane.

In this paper, we study the security in the data plane. First, we introduce freeloading, a novel SDN-based attack allowing attackers to send their malicious packets by exploiting the existing data plane. Through freeloading, attackers can avoid the following: the risk of exposure to the controller and the network operators (which, for instance, can facilitate anomaly detection by traffic monitoring), the costs of processing delay and the communication overhead (which are incurred from establishing the data plane and the traffic paths).

To defend against freeloading, we propose a flow watermarking technique to protect the origin integrity of the forwarding path. Each legitimate host can embed its own secret message (a watermark) into the payload of any packet. The unique watermark is used as a signature of ownership of flow rules in the data plane since the watermark can only be generated by the source host who initiated the communication by sending the first packet to the data plane and established the flow rule with the controller. The destination host can verify the unique watermark by decoding it based on preshared secret configuration information. Any packet that does not contain this watermark signature is not authorized to use the data plane and will be dropped.

This paper makes four important contributions to construct reliable and secure SDN. First, we address the misuse of the data plane in SDN and present freeloading. Second, we propose the watermarking technique to defend against the freeloading attack. Lastly, we implement and experiment our proposed method to show its effectiveness.

This paper is organized as follows. Section II will explain SDN in general and introduce the new freeloading attack. Section III will present our proposed methods, watermarking. Experimental results will be discussed in Section IV. Section V will review the related literature. Finally, we will conclude our work suggesting directions for future research in Section VII.

II. PROBLEM STATEMENT

The history of computer network, most notably the Internet, has demonstrated that users inject unauthorized packets for variety of purposes. The users can be selfish (such as stealthily piggybacking on other users’ network resources to minimize costs) or malicious (such as DoS attacks for remote servers, malware propagation into network, spam, and exploitation of remote hosts). We collectively call them attackers and introduce a novel threat that facilitates their operations.

To motivate our attack, we first focus on the attacker’s point of view. As discussed in Section II-A, all users need to establish

*The first two authors equally contribute to this work and the first author is a corresponding author. This work is supported by the Research Foundation of San Jose State and by the research grant for the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore’s Agency for Science, Technology and Research (A*STAR).
the path access to utilize the forwarding path; in SDN, this process involves the interaction with the network controller. A naïve attacker will comply to such protocol and assume the overhead in control communication with the controller (which can hinder the timely actuation of the attack) and risk being vulnerable to reactive security solutions implemented by the controller (such as traffic monitoring and source detection).

We consider a smarter attacker. In our threat model, the attackers exploit existing flow rules to launch malicious actions instead of installing new flow rules themselves, making the attackers unaccountable for their behaviors and the detection difficult. They could scrounge for existing flow rules in a flow table while intercepting any communication between two hosts. We call such an SDN attack freeloading. Such an attack is very difficult to detect and prevent because (1) the attack traffic is piggybacking on the existing flow rules used by benign hosts and (2) the flow establishment does not involve interactions with the network controller. In other words, the attack traffic is not processed through the controller, which generally assumes the security implementation and enforcement responsibilities in SDN. Furthermore, since such attacks avoid the installation of new flow rules for attackers, it enables attackers to bypass the installation cost of new flow rules.

![Figure 1. An illustrated freeloading attack in a virtual network in Mininet.](image)

### A. Freeloading Attack: An Instantiation

Freeloading is possible because the current SDN designs and practice do not offer mechanisms to verify who is using flow rules beyond the address matching. To launch a freeloading attack, the attackers need to be aware of the flow rules. The network-layer metadata, including flow rules, can be easily obtained via packet capture. In addition, there are well-studied techniques to get the information of flow rules in the data plane [15], [17], [35]. In other words, the attackers can utilize the current network topology information [35] and the timing information because of the processing delay [15], [17] (for instance, the flow rule establishment takes greater time than when the flow rule is already established and the packet forwarded). The attackers can also get the current network information by using various networking tools such as ping, traceroute, LLDP (Link Layer Discory Protocol). In addition, man-in-the-middle attacks and spoofing attacks can be used in order to get the flow information in SDN [36].

Assuming that the attacker physically compromised the local network, the attacker first sniffs neighboring hosts to acquire necessary data such as the source and destination IP/MAC addresses and the communication port numbers. Then, it sends its spoofed packets using the information obtained and launches stealthy attack without installing flow rules. Figure 1 illustrates such process with Host 1 communicating to Host 2. Host 3 targets the neighbor host, Host 1 for sniffing. When Host 1 initiates its communication with Host 2 (i.e. a victim host), which process involves exchanging ARP request/response packets with Host 2 and establishing flow rules with the controller, Host 3 can obtain the necessary information of Host 1 and Host 2. Afterward, Host 3 can make its spoofed packets by using the sniffed information and then send unauthorized packets to Host 2 without installing the flow rule itself.

### III. Our Proposed Countermeasure

This section presents our proposed method to defend against the freeloading attack that was presented in the previous section. To detect the misuse of flow tables in the data plane, our proposed defense method is watermarking consisting of the encoding and the decoding system. The proposed watermarking technique is to identify the origin of packets for detecting malicious packets by checking the ownership of the used flow rules in the data plane. It involves the cooperation between the controllers, the switches, and the destinations.

#### A. System Architecture

We assume a priori established key between the users, for example, by using public-key infrastructure (PKI) and offline certificate authories (CA); such assumptions are typical in conventional watermarking techniques [20], [29] and, more broadly, in general secure protocol design. We leverage such infrastructure to share the watermark configuration information between a source host and the verification host, for example, the centralized controller securely distributes the configuration information to the source and the destination during the bootstrapping process, e.g., when the flow rule is established. Secure distribution of keys and configurations [39], [40] is beyond the scope of our contribution. In addition, the self-synchronization method during the decoding process can be applied to avoid sharing configuration information in advance as presented in [29].

The encoding/decoding system can be placed as a wrapper function to capture packets between the application layer and the transport layer based on TCP/IP protocol stacks. The watermark will be embedded into application data (i.e. payload) in the packet through socket functions. Since many IP packets have a full payload and it is difficult to find space inside the payload of IP packets, the watermark is processed at the application layer. Thus, our scheme is robust against obfuscation-based threats that occur at the transport layer such as chaff, repacketization, flow splitting, and so on [29]. This is in contrast to the traditional watermarking techniques to find the origin of attacks through intermediate hosts in the network.

When a source host sends traffic after gaining legitimate access by having the controller establish the flow rules on the switches, the encoding system on the source host embeds a watermark into the payload according to predefined configurations. The decoding system (which can be located at the destination host or at the controller) verifies whether the
source host uses its own flow rules installed between a source and a destination. If the decoded watermark does not match with the pre-shared configuration, the destination host sends out an alert to disclose the misuse of the flow rules to the controller.

B. Encoding and Decoding a Watermark into Data Payloads

We first describe the watermark generation. The configuration information, which is shared between the source host and the decoding host as described in Section III-A, includes a watermark \( f \), the length of the watermark \( l \), the random position to embed the watermark bits into the data payload \( d \). A traffic flow \( F \) with a series of packets \( F = \langle P_1, P_2, ..., \rangle \) is in order of the departure time from a source host. Per configuration, the watermark is embedded in a packet or across multiple packets. Given an encoding or decoding function and the watermark \( FPT = \langle f, l, d, k \rangle \), the encoding system embeds the intended watermark \( f \) into the payload for a given flow \( F \). The last value \( k \) is a binary 0 or 1 to enable or to disable the option of extending the watermark across multiple packets. In other words, if \( k = 0 \), the encoding system encodes all the watermark bits into the payload of one packet according to the random position \( d \). The watermark bits are always a multiple of eight starting from 24 bits. For encoding multiple packets, the number of packets needed is derived by dividing the total watermark bits by eight. Encoding the watermark bits is based on one byte (8 bits) since the packet is formatted in 8-bit bytes with control bytes.

![Encoding a fingerprint](figure.png)

Figure 2. An illustration of encoding a 24-bit watermark into one packet. (Note that \( f = 100110110011001101110011001011000111101111100110011011100110010110001 \) for \( FPT = \langle f, l, d, k \rangle \)).

Fig. 2 illustrates the encoding of a 24-bit watermark when \( k = 0 \). The watermark bits shown in red are divided into 3 groups of 8 bits. According to the random distance \( d \), each group is embedded after 1 byte, 3 bytes, and 5 bytes offset, as shown in Fig. 2. Because \( k = 0 \), only one packet is used to encode the 24-bit watermark. If \( k = 1 \), three packets would be needed to embed each of the 8 bits into different packets. Specifically, the first packet would have the first 8 bits of \( f \) after 1 byte offset in the payload of the first packet. The second packet would have another 8 bits of \( f \) after 3 bytes offset, starting from the first byte of the payload. Finally, the third packet would have the last 8 bits of \( f \) after 5 bytes offset in the payload of the third packet. This encoding rule can be changed by using different configuration information.

The encoding process on a source host is simply performed when applications generate user data (payload) to be sent to a destination. The process is activated after a flow rule is installed on the switch. Because free-lacing is launched on the existing flow rules, our proposed method encodes a watermark after the first successful communication (i.e. after installing flow rules). In addition, depending on the network protocol, the packet size varies from 7 to the maximum 65,542 bytes including the packet header. Some packets do not have payload like SYN or ACK, in which case it is not necessary to use an encoding system because it might not be possible to be exploited by a freeloader. After SYN packets, a flow rule must be actually installed to prevent any possible DoS attack, as addressed in [17]. ACK is also often piggybacked with user data instead of a stand-alone packet. Therefore, any packet can be encoded with a watermark after flow rule installation.

Similarly, the decoding system with the secret configuration information \( FPT = \langle f, l, d, k \rangle \) used to encode \( f \) is now able to decode \( f' \) and match against \( f \). Before reading user data, the decoding system deletes the embedded watermark from the payload according to the distance \( d \). Given a watermarked flow \( F' \) and shared secret information \( (f, l, d, k) \), the decoding system is able to decode a watermark \( f' \) and then compare it against \( f \) for detection purposes. If the original watermark \( f \) is matched with the decoded \( f' \), we can verify that the installed flow rule was used by the original flow rule requester, not an attacker. If not matched, the decoding system can drop the packets and inform a controller of the detection.

A Hamming distance threshold \( h \) is used to filter out any noise in the process, including active transformations of the payload by attackers. Let \( H(f', f) \) represent the Hamming distance between \( f' \) and \( f \). If \( h(0 \leq h \leq l) \) is a decoding threshold, the decoding system reports a positive detection. Although a higher threshold increases the detection rate (the possibility of correctly identifying a flow as being encoded with \( f \)), it also increase the false positive rate (the possibility of falsely identifying a flow as being embedded with \( f \)). The computation of the expected detection rate and the expected false positive rate for an \( l \)-bit watermark using a Hamming distance threshold \( h \) was shown in [20]. It is suggested that \( h \) be chosen to balance the tradeoff between the detection rate and the false positive rate.

If attackers launch a freeloading attack, the packet will not have the correct watermark in the payload. Since the attackers do not have the configuration information, they cannot generate the same source watermark. Thus, given a packet, the watermark presence provides a strong assurance for the legitimacy of the packet. In the next section, we describe which packets get inspected, i.e., for greater efficiency than inspecting all packets, how we choose a subset of the packets in a flow to apply the watermark verification; we use traffic anomaly detection to provide more accurate selection than the naive random selection.

IV. Evaluation

In this section, we explain our implementation and the experimental setup, simulate various attacker strategies, and demonstrate the effectiveness of our proposed solutions.

A. Implementation and Experiment Setup

We implemented the proposed method based on the socket library to read packets and to embed a watermark into packets. We also implemented the python PCAP file library to check the embedded watermarks by reading packets. Fig. 1 shows a virtual network created in Mininet 2.1.0 based on Open vSwitch.
number of flow rules

Host

Similarly, launched from Host

of the new installed flow

security monitor tools on the controller to monitor the activity

port numbers.

packets

real networks

expected that the delays takes effect more than these values in

to install one flow rule, it requires around 0.5 ms to launch these

more than 200 flow rules were installed. If we consider the time

the overhead in

same

of flow rules installed in the other switch (S2) was almost the

the attackers

attacks

setting using actual computers as hosts.

Fig. 2 shows the results of port scan attacks and SYN flood

attacks in the SDN. Fig. 2 shows the number of flow rules that

the attackers installed in the switch (S1) over time. The number

of flow rules installed in the other switch (S2) was almost the

same as in S1. In addition, the figure demonstrated the time

overhead in launching attacks in the SDN environment due to

the flow rule installation time. For example, during 10 seconds,

more than 200 flow rules were installed. If we consider the time
to install one flow rule, it requires around 0.5 ms to launch these
attacks in Mininet, but more than 4 ms in the emulatrion. We

expected that the delays takes effect more than these values in
real networks. This is to be expected since each attack requires
the installation of many new flow rules to forward attack
packets. Both attacks required installation of almost the same
numbe of flow rules per packet with respect to applications or
port numbers. Moreover, the flow installation will enable the
security monitor tools on the controller to monitor the activity
of the new installed fow rules.

First, in the case of a port scan attack, the attack was
launched from Host 1 to Host 2 to scan any port of Host 2. In
SDN in Mininet, the controller enforced both of the switches
(S1 and S2) to install every single flow rule for each packet
because each packet had targeted different port numbers.
Similarly, a SYN flood attack was performed from Host1 to
Host 2 on Host 2’s ports 22 and 80. As the attack started, the
number of flow rules increased over time. Therefore, due to the
flow rule installation time, the time to lunch attacks also
increased as the number of packets increased.

![Figure 2. The number of flow rules in the switch installed by attacks over time.](image)

**Freeloading attacks**: We evaluated a freeloading attack over time. From Fig. 1, Host 3 caused a freeloading attack to Host 2 by using the flow rules that Host 1 was using. As a result, we confirmed that the response of Host 3 went back to Host 1 instead of Host 3. In addition, we found that there was no difference in the number of flow rules installed in switch S2 as shown in Fig. 3. In other words, even though Host 3 sent a new packet to Host 2, Host 3 did not need to wait to install flow rules in both switches, S1 and S2. It just took some time (0.005 ms on average) at first to get the target host information, but it eventually consumed a very small amount of time to launch the malicious actions in SDN in order to understand the existing flow rules. This was expected since the freeloading attack just utilized existing flow rules already installed in the switches. It doesn’t require to install any flow rule to send any packets. Therefore, no communication and processing cost are not involved between the controller and the data plane. Compared to the results of the two well-known attacks, freeloading can be launched with existing flow rules on the switches, with the very small delay time at first. Furthermore, this attacks can not be monitored by the controller because the malicious traffic is injected into the benign traffic on the existing flow rules. This is a serious attack that circumvents the fundamental requirement of SDN for installation of new rules and avoid security monitoring on the controller.

Fig. 3 shows the difference of installed flow rules with or without a freeloading attack. Except for the number of packets (i.e. \( n \_\text{packets} \) in the flow rule) in order to report statistics to the controller, the installed flow rules were not changed at all. The important observation is that a new flow rule was not installed despite of the first new packet between attacking host (Host 3) and Host 2 in Fig 1. It is not possible to distinguish freeloading packets based only on the number of the packets. To prevent such attacks, we proposed a new sophisticated defense method, watermarking. It can also provide a reliable end-to-end communication between a source and a destination. It will be our future work to develop various useful applications based on watermarking.

On the other hand, depending on the attack type, the time needed to launch a freeloading attack varies. For example, a port scan attack might need a long time because attackers need to wait for new open ports of target hosts. By contrast, DoS
service attacks and malware propagation attacks might take a much shorter time since existing rules can be used by using a local network topology information. The most important fact is that attackers can send their malicious traffic based on the installed flow rules by using various existing network tools and network-based attacks. In addition, freeloading might be very difficult to detect because it loads their traffic on benign traffic without installing any flow rules.

Through this type of freeloading attacks, SDN has damaged its functionality. First, the flow rule installation requirement of SDN was broken even though the attacker send a first new packet in the network. Second, attacker can mix their malicious flows into the existing benign flows in order to hide the attacking flow. Third, the installation of new flow rules for attackers provide enough information so that the controller can monitor the traffic behavior in given statistic data from the data plane. However, this attack can prevent the controller from monitoring their behaviors. Lastly, once attackers get the data plane information, they can process their malicious packets faster. It is because the communication cost and the process cost is small between the control and the data plane.

Robustness and effectiveness of watermarking: To evaluate the robustness of the proposed method, we actively changed some values in the payload according to attack rate. The attack rate will be from 0% to 50% to obfuscate the percentage of the payload. For example, if the attack rate is 20%, only 20% of the total payload per packet will be randomly changed. To defeat the watermarking technique, attackers might develop other obfuscation techniques, but we have only assumed this simple randomness to alter the payload of a packet.

As shown in Fig. 4, the detection rate of the watermark was almost 98% for attack rates up to 50% with configurations as shown in Fig. 4. For attack rates up to a 30% attack rate, the detection rate of the watermark was 100% for all the configuration values used in these experiments. This shows how difficult it is for attackers to change the watermark, since they do not have any knowledge of the configuration values. In the case where multiple packets are encoded, it is harder for attackers to obfuscate payloads because the watermark will be spread throughout different packets randomly. They can only randomly inject bytes into packets on the fly, which requires a high level of technical skill.

Figure 3. The comparison of flow rules in a switch (S2) with or without freeloading attacks.

(a) Flow rules before a freeloading attack
(b) Flow rules after a freeloading attack

Figure 4. A detection rate under various attack rates.

(a) One packet used for encoding and decoding
(b) Multiple packets used for encoding and decoding
incoming packets since we used the collected dataset based on including reading cores, 2.27GHz and 4 GB RAM on 64 bit Ubuntu 12.04, multiple packets. This was tested on Intel Core i3 with quad watermark packets when we choose depending on the random distance bit bytes with control bytes the total \( l, d, k > \) watermark effectiveness of the large value can theoretically be used as in [20]. Therefore, the threshold value should be less than 7 even though highe.

The number of packets to embed a watermark affects the effectiveness of the watermarking scheme. It is proportional to the length of the watermark. Comparing to traditional watermark schemes [20], [29], this new watermark technique doesn’t require many packets since it embeds a watermark into data payloads instead of using packet timings. Given \( FPT = \frac{i}{f, l, d, k} \), the number of packets needed is derived by dividing the total watermark bits by eight. Encoding the watermark bits is based on one byte (8 bits) since the packet is formatted in 8-bit bytes with control bytes. To embed a 48-bit watermark, depending on the random distance \( d \), we might need around 6 packets when we choose \( k=1 \). If \( k=0 \), it is better to choose a 24-bit watermark to use only one packet.

Table 1 shows encoding and decoding times for different watermarks depending on whether there is one packet or multiple packets. This was tested on Intel Core i3 with quad cores, 2.27GHz and 4 GB RAM on 64 bit Ubuntu 12.04, including reading packets, converting hex values, and encoding \( f \) according to the distance \( d \). There is no waiting time for incoming packets since we used the collected dataset based on PCAP files. The decoding time was bigger than the encoding time since it needs to find a right configuration file for decoding a right watermark under the same procedure. To reduce the decoding time, when the communication between sources and destinations is established, the preconfiguration information can also set up through a transport security in SDN.

V. RELATED WORK

Many prior work has successfully implemented SDN. FlowVisor has proposed a virtual network by slicing the network and has provided an modular platform for OpenFlow controllers [14]. VeriFlow has presented a method for checking invariant property violations in the network through network slicing [9]. FleXam has suggested a sampling method that enables the controller to access packet-level information [18]. While the aforementioned work has focused on the general virtualization aspect of SDN, others have focused more on the routing behavior of SDN. FortNox enables NOX to check contradiction of flow rules in real time, and safeguards against malware applications that may update rules to contradict existing flow rules [13]. FLOWER has proposed a model checking system to verify the compliance of a flow rule set against an invariant security policy [19]. Our work operates orthogonally to these work; while these work investigates the flow rule itself to detect anomaly, our work embeds watermark and implements the origin integrity at the application layer, i.e., the watermark is embedded at the data payload.

In OpenFlow, which enables communication between the controller and the routing switches, poor rule design can lead to accidental saturation of queries to the controller, creating a bottleneck in all switches because of the heavy reliance on the controller. Researchers have highlighted this vulnerability and its security implications in prior work [11], [15], and others have proposed solutions in traffic monitoring [4] and rate-limiting [11]. In contrast to these prior work which adopts reactive solutions, we offer a proactive approach in protecting the source integrity of flow rule.

Others have also used machine learning techniques in the security context to detect abnormal flows by selecting important features from network flows [31], [32]. Livadas et. al [31] and Narang et. al [32] detected botnet traffic based on selecting important features based on Naïve Bayes algorithm and the Bayesian Network classifier. Braga et. al. proposed the use of machine learning techniques for DDoS flooding attack detection based on self organized maps in NOX/OpenFlow [4]. These prior work extracted more feature sets from the flow table than we did; we can incorporate their techniques into our work to improve accuracy, as we will discuss in Section VI.

VI. CONCLUSION

This paper introduces and studies the feasibility of freeloading attack. It also presents a countermeasure to detect such attacks and to identify the misuse of flow table. Our work is the first to address the potential for a flow rule (installed on a switch) to be exploited by attackers for malicious purposes. To defend against such attacks, we have proposed a scheme that embeds watermarks on the packet payload, which can then be used to detect the misuse of flow rules when the rules are not used by unauthorized users. Our proposed method is resilient against attackers who obfuscate the watermarking signature and
performs well in a noisy environment with high detection rate and low false positive rate.

REFERENCES


[23] R tool http://www.r-project.org/


