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
sychg@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 routing. 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 Vswtich, 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 [23], [24]. 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 [2], and others [3], [15], [16], [30]. 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 watermark by decoding it based on pre-shared secret configuration information. Any packet that does not contain this watermark signature is not authorized to use the data plane and thus will be dropped.

This paper makes three 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 STATMENT

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 introduce a novel threat that facilitates such operations and call the entity subjects *attackers*.

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 path access to utilize the forwarding path; in SDN, this

*The first two authors equally contribute to this work and Professor Younghee Park is the 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).

process involves the interaction with the network controller. A naïve attacker who wants to access the network 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 subjected to reactive security measures implemented by the controller (such as traffic monitoring and source detection).

We consider an intelligent 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, it enables attackers to bypass the installation cost of new flow rules.

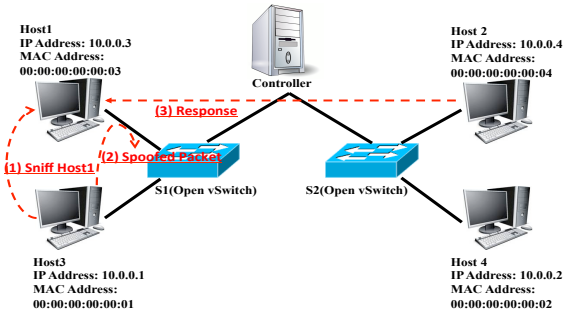


Figure 1. An illustrated freeloading attack in a virtual network in Mininet.

A. *Freeloading Attack: An Instantiation*

Freeloading is possible because the current SDN practice does not offer mechanisms to verify who is using flow rules beyond the address matching. To launch freeloading, the attackers need to be aware of the flow rules. The network-layer metadata, including flow rules, can be easily obtained via packet capture or other more sophisticated techniques in the data plane that uses the network topology information [35] or timing information due to processing and the control-communication-overhead delay [15], [17]. The attackers can also get the current network information by using various networking tools such as ping, traceroute, LLDP (Link Layer Discovery Protocol). In addition, man-in-the-middle attacks and spoofing attacks can be used to get the flow information [36].

After compromising the physical access to the local network, the attacker first sniffs neighbor 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. When Host 1 initiates its communication with Host 2 by exchanging ARP request/response and establishes flow rules with the controller, Host 3 can first obtain the necessary information of Host 1 and Host 2 and then use that information to send unauthorized packets to Host 2 without installing the flow rule itself.

III. Our Proposed Coutermeasure

This section presents our proposed method to defend against the freeloading attack. To detect the misuse of flow tables, our proposed defense method is watermakring-based encoding. The watermark can help to identify the origin of packets for detecting malicious packets and check the ownership of the used flow rules. It requires 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 certificate authorities (CA); secure distribution of keys and configurations [23], [24] is beyond the scope of our contribution, and such assumptions are typical in conventional watermaking [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 distributes the configuration information to the source and the destination during the bootstrapping process, e.g., when the flow rule is established. In addition, the self-synchronization during decoding can be applied to avoid sharing configuration information in advance [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, re-packetization, 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 $F = <P_1, P_2, ... >$, is indexed 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 $FPT = < f, l, d, k >$, the encoding system embeds the intended watermark *f* into the payload for a given flow *F*. The last value *k* is a control bit to disable the option of extending the watermark across multiple packets. If *k* is 0, the encoding system encodes all the watermark bits into the payload of one packet in the positions specified by *d*. The watermark bits are multiples 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.

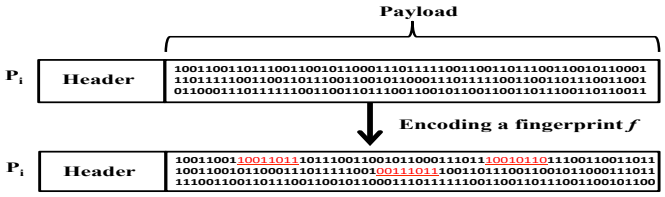


Figure 2. An illustration of encoding a 24-bit *watermark* into one packet. (Note that *f* = 100110111001011000111011, *l*=24, d={1, 3, 5}, *k*=0 for *FPT* = < *f*, *l*, *d*, *k* >.)

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* is 0, only one packet is used to encode the 24-bit watermark. If *k* were 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 freeloading 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. 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 $FPT = < f, l, d, k >$ used to encode *f* is now able to decode a *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^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 $H(f', f) \le h$, where $h(0 \le h \le l)$ 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 increases the false positive rate (the possibility of falsely identifying a flow as being embedded with *f*). The decision on *h* affects the tradeoff between the detection and the false positive [20].

The watermark presence provides a strong assurance for the legitimacy of the packet, as the *unauthorized* attackers do not have the configuration information by definition and cannot generate the correct watermarks.

IV. Evaluation

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. We constructed a virtual network in Mininet 2.1.0 based on Open vSwitch 2.0.0 version on Linux [1]. Mininet creates customized topology with OpenFlow switches, routers, and the controller. As described in Figure 1, the experimental setup consisted of one SDN controller, two OpenFlow-enabled switches and four hosts in a network. We used POX for the SDN controller using OpenFlow 1.0, which is the default controller in Mininet. To generate flows and to replay PCAP files collected from public websites, we used Tcpreplay [21] on the Mininet. In addition, to generate attacks on the Mininet, we used Scapy [22].

We simulate threat scenarios described in Fig. 1. First, we evaluated a port scan attack and a SYN flood attack to measure the major characteristics of these two attacks in SDN. We also analyzed the time required to launch a *freeloading attack* by using an IP spoofing attack in the SDN environment. Lastly, we evaluated the effectiveness of the proposed method in detecting the watermarks under various attacks.

B. Experimental Results

Attacks without freeloading: To motivate freeloading, we first study the naïve attacker's overhead when it attacks the network without taking advantage of the freeloading vulnerability. (As discussed previously, our work is the first to build integrity and check the enforcement of flow rule establishment.) We first tested the processing time to install one flow rule on swithces. It takes, on average 0.0024 ms for a switch to install one flow rule on the switch S1 in Mininet-based simulation, and the same process requires 0.0167 ms for emulator-based testbed using actual computers as hosts.

Fig. 2 shows the results of port scan attacks and SYN flood attacks in the SDN and 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) is comparable to that of S1. In addition, The figure demonstrated the time overhead in launching attacks due to the flow rule installation time. For example, it requires about 0.5 ms to launch these attacks in Mininet and more than 4 ms in the emulation. We expected that the delays takes effect more than these values in real networks since each attack requires the installation of many new flow rules to forward attack packets. Both attacks required installation of almost the same number 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 installed flow rules.

First, in for port scanning, Host 1 acted as an attacker and scanned the ports 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.

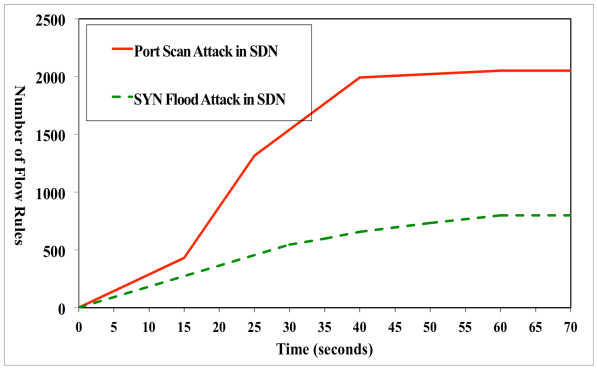
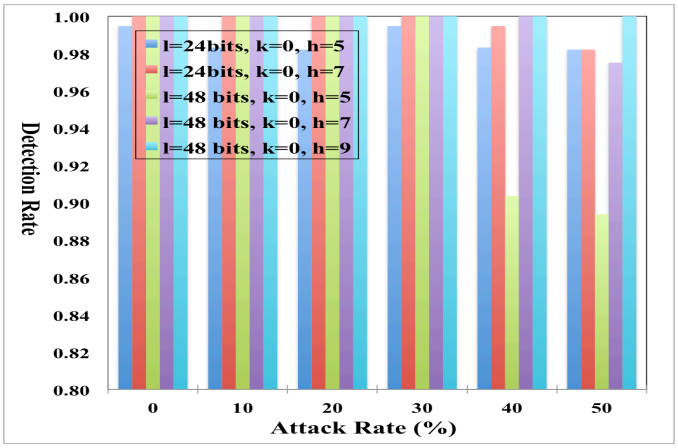


Figure 2. The number of flow rules in the switch installed by attacks over time (Freeloading attack resulted in no flow rules installed over time and is thus omitted here).

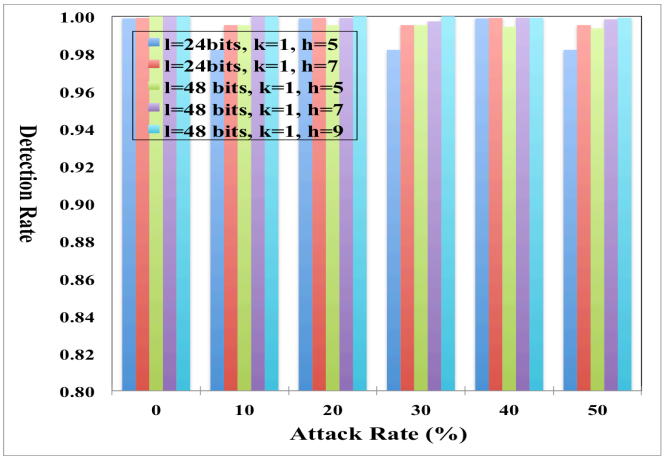
Freeloading attacks: We evaluated a *freeloading attack* over time. From Fig. 1, Host 3 caused a *freeloading attack* to Host 2 by using the flow rules that Host 1 set up. As a result, the network held Host 1 accountable when Host 3 used the flow. In addition, there was no difference in the number of flow rules installed in switch S2. 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 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 costs are 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.

At a per-packet level, there was no difference between the legitimate packets and the freeloading packets. Although a flow traffic analyses (intuitively, the number of packets do increase) can be used for attack detection, but this will likely penalize the legitimate user Host 1 as well since the two flows cannot be distinguished from each other. We introduce watermarks to distinguish the legitimate and the freeloading packets. Further developments such as an end-to-end security leveraging the watermark is beyond the scope of this paper.

Robustness and effectiveness of watermarking: To evaluate the robustness of the proposed method, we varied 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.

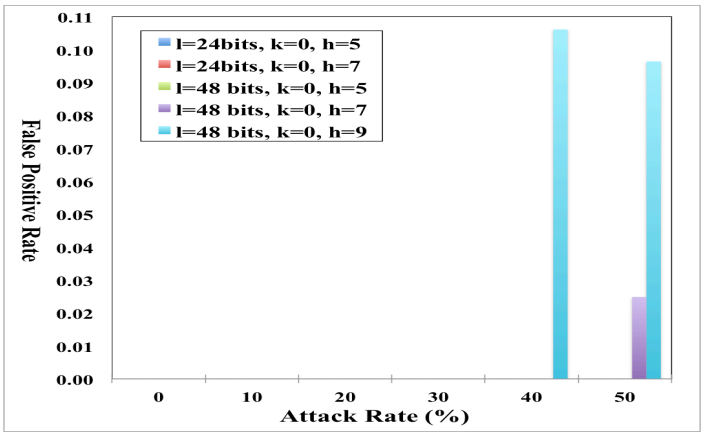


(a) One packet used for encoding and decoding

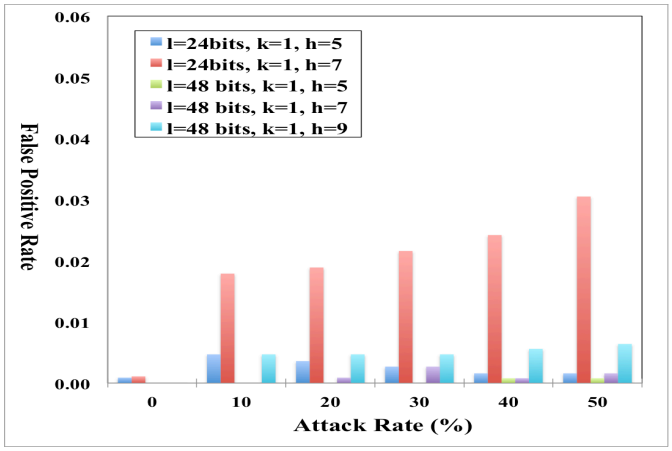


(b) Multiple packets used for encoding and decoding

Figure 3. A detection rate under various attack rates.



(a) One packet used for encoding and decoding



(b) Multiple packets used for encoding and decoding

Figure 4. A false positive rate under various attack rates.

As shown in Fig. 3, the detection rate of the watermark was almost 98% for attack rates up to 50% with configurations as shown in Fig. 3. 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 great level of sophistication and increases the attacker cost.

Fig. 4 showed false positive rates according to various attack rates since the detection rate was almost 98% at any attack rates. Fig. 4 demonstrated the robustness and the effectiveness of the proposed method even when attackers try to obfuscate the payload. In Fig. 4 (a) and (b), the false positive rate was less than 0.037 % whether using one packet or multiple packets to encode a 24-bit watermark into the payload. Compared to 48-bit watermark with the same values as other parameters, the false positive rate dropped to 0.024%. However, if we use the decoding threshold $h=9$, the false positive rate increased up to 0.1% in spite of a 48-bit watermark .

Throughout the experiments, the embedded watermark was strong enough to identify whether the payload was generated by the installed flow rules from a particular source. The watermark was not sensitive to an attacker's obfuscation, which failed to defeat the watermarking method. In fact, other intrusion detection systems as well as our proposed system can identify if attackers change the entire original payload. In addition, the higher the threshold value (h), the higher the false positive rate. Therefore, the threshold value should be less than 7 even though the large value can theoretically be used as in [20].

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 $FPT = < f, l, d, k >$, 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.

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]. FLOVER 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 studies the freeloading vulnerability and 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 evaluations show that our proposed method is resilient against attackers who obfuscate the watermarking signature and performs well in a noisy environment.

References

[1] Open vswitch. http://openvswitch.org.

[2] Sungmin Hong, Lei Xu, Haopei Wang, Guofei Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures." In Proc. of 22nd Annual Network & Distributed System Security Symposium (NDSS'15), San Diego, CA, USA. February 2015.

[3] Antikainen, Markku and Aura, Tuomas and Särelä, Mikko, "Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch", Secure IT Systems, Lecture Notes in Computer Science, pp229-244, Springer, 2014.

[4] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito, " Lightweight ddos flooding attack detection using nox/openflow," In Proc. of IEEE Conference on Local Computer Networks, LCN, 2010.

[5] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An api for application control of sdns," SIGCOMM Comput. Commun. Rev., 43(4):327–338, 2013.

[6] S. K. FG¨ottingen, oayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions," In Proc. of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking(HotSDN), 2013.

[7] O. N. Foundation. Software-defined networking: The new norm for networks. Technical report, 2012.

[8] R. Hand, M. Ton, and E. Keller, "Active security," In Proc. of ACM Workshop on Hot Topics in Networks, HotNets-XII, 2013.

[9] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time,". In Proc. of ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), 2012.

[10] R. Kloti, V. Kotronis, and P. Smith. Openflow: A security analysis. In Proc. of IEEE International Conference on Network Protocols, ICNP, 2013.

[11] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In Proc. of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, 2013.

[12] OpenFlow. Openflow switch specification version 1.0.0. Technical report, 2010.

[13] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In Proc. of ACM Workshop on Hot Topics in Software Defined Networks, HotSDN '12, 2012.

[14] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In Proc. of USENIX Conference on Operating Systems Design and Implementation, OSDI'10, 2010.

[15] S. Shin and G. Gu. Attacking software-defined networks: A first feasibility study. In Proc. of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, pages 165–166, 2013.

[16] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security service for software-defined networks. In Proc. of Annual Network and Distributed System Security Symposium, NDSS 2013.

[17] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in softwaredefined networks. In Proc. of ACM SIGSAC Conference on Computer and Communications Security, CCS '13, pages 413–424, 2013.

[18] S. Shirali-Shahreza and Y. Ganjali. Flexam: Flexible sampling extension for monitoring and security applications in openflow. In Proc. of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), 2013

[19] S. Son, S. Shin, V. Yegneswaran, P. A. Porras, and G. Gu. Model checking invariant security properties in openflow. In Proc. of IEEE International Conference on Communications, ICC, 2013.

[20] X. Wang and D. S. Reeves, "Robust Correlation of Encrypted AttackTraffic through Stepping Stones by Manipulation of Interpacket Delays," in Proc. of ACM conference on Computer and CommunicationSecurity (CCS), Oct. 2003.

[21] Tcpreplay. http://tcpreplay.synfin.net/

[22] Scapy. http://www.secdev.org/projects/scapy/

[23] J. Chiang and Y.C. Hu, "Dynamic Jamming Mitigation for Wirelss Broadcast Networks," in Proc. of IEEE Conference on Computer Communications (Infocom), Phoenix, AZ, April 2008.

[24] S.Y. Chang, Y.C. Hu, and N. Laurenti, "SimpleMAC: A Jamming-Resilient MAC-Layer Protocol for Wireless Channel Coordination," in Proc. of ACM Conference on Mobile Computing and Networking (MobiCom), Istanbul, Turkey, August 2012.

[25] Mitchell, Thomas M., "Machine Learning," McGraw-Hill, Inc., New York, NY, USA, 1997.

[26] S. Mousavi, "Early Detection of DDoS Attacks in Software Defined Networks Controller," Master Thesis at Carleton University, May, 2014

[27] Open Networking Foundation. "Software-Defined Network: The New Norm for Networks," ONF White Paper, April, 2012.

[28] T. Koponen et al., "Onix: a distributed control platform for large-scale production networks," In Proc. of USENIX Conference on Operating Systems Design and Implementation (OSDI), 2010.

[29] Xinyuan Wang, Shiping Chen and Sushil Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. In Proc. of the 2007 IEEE Symposium on Security & Privacy (S&P 2007), May 2007.

[30] Scott-Hayward, S., O'Callaghan, G. and Sezer, S, "SDN Security: A Survey," In Proc. of IEEE SDN for Future Networks and Services (SDN4FNS), 2013.

[31] C. Livadas, R. Walsh, D. Lapsley, W. Strayer, "Using machine learning technliques to identify botnet traffic," in Proc. of IEEE Conference on Local Computer Networks, 2006.

[32] Pratik Narang, Jagan Mohan Reddy and Chitterranjan Hota, "Feature Selection for Detection of Peer-to-Peer Botnet Traffic," in the Proc. of ACM Computing Convention, 2013.

[33] Young June Pyun and Dogulas S. Reeves, "Strategic deployment of network monitors for attack attribution," in the Proc. of IEEE International Conference on Broadband Communications, Networks and Systems (BROADNETS), 2007.

[34] Xiaosong Wang; Li Wang; Benhai Yu; Guixue Dong, "Studies on Network Management System framework of Campus Network," In Proc. of 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR), March 2010.