American Journal of Networks and Communications
2016; 5(2): 27-34
http://www.sciencepublishinggroup.com/j/ajnc
doi: 10.11648/j.ajnc.20160502.13
ISSN: 2326-893X (Print); ISSN: 2326-8964 (Online)

Combat-Sniff: A Comprehensive Countermeasure to Resist Data Plane Eavesdropping in Software-Defined Networks

Fan Jiang, Chen Song*, Hao Xun, Zhen Xu

Institute of Information Engineering, Chinese Academy of Sciences, Beijng, China

Email address:

jiangfan@iie.ac.cn (Fan Jiang), songchen@iie.ac.cn (Chen Song), xunhao@iie.ac.cn (Hao Xun), xuzhen@iie.ac.cn (Zhen Xu)

*Corresponding author

To cite this article:

Fan Jiang, Chen Song, Hao Xun, Zhen Xu. Combat-Sniff: A Comprehensive Countermeasure to Resist Data Plane Eavesdropping in Software-Defined Networks. *American Journal of Networks and Communications.* Vol. 5, No. 2, 2016, pp. 27-34.
doi: 10.11648/j.ajnc.20160502.13

Received: February 23, 2016; **Accepted**: March 4, 2016; **Published**: April 21, 2016

Abstract: Software-defined networking (SDN), on account of its unprecedented capability of network traffic monitoring and data resource transferring, has been deployed into a wide range of application scenarios. However, typical cyber-attacks which prevail in traditional IP networks, have also mutated their implementation models adjusting to SDN environment. Eavesdropping is one of such attacks and causes severe information disclosure to different degree. In this paper, we focus on data plane eavesdropping in SDN and treat it on two levels according to the extent an adversarial sniffer can exploit a SDN switch. Then we introduce Combat-Sniff, a comprehensive countermeasure which includes two methods to deal with the two-level sniffing respectively. And later, we both theoretically and experimentally demonstrate their reliability and performance. Results represent that we can exert Combat-Sniff in SDN to satisfy different security requirements with an acceptable overhead.

Keywords: Eavesdropping, Software-Defined Networking (SDN), Flow Entries Integrity Verification, Moving Target Defense (MTD)

1. Introduction

SDN has offered traditional IP networks a brand new paradigm [1], with its separated control and forwarding planes, logically centralized controllers, and unified programmable interfaces. Since the original rigid closed network model suffers a hardship when keeping pace with the rapid expansion of network size and abrupt outburst of huge data volume, newly developed network practices, such as cloud service and big data analytics, turn to SDN for feasible solutions. Innovative as SDN infrastructure is, it fails to put an end to traditional typical cyber-attacks, which have exploited their distinctive realization methods against SDN-specific background. As the present mostly referenced implementation of SDN is OpenFlow protocol [2], we would focus our subsequent discussion on OpenFlow-based network.

Network eavesdropping [3] is a kind of packets interception attack in traditional IP network. So far, there are no recognized reliable detection methods to deal with it, and the accepted defense method is encryption [4]. However, situation changes when transiting to SDN, either for detection or defense. In this paper, we focus on data plane eavesdropping in SDN. According to what degree a malicious attacker can exploit a SDN switch, in figure 1, we classify the eavesdropping into two levels: flow entries compromised level and switch compromised level.

Flow entry compromised level. The decoupled control and data plane expose a forwarding rule inconsistency problem. Central controllers take charge of networking intelligence by means of installing flow entries on flow tables in switches to instruct traffic forwarding. Maliciously falsifying flow entries from switch side can cause the inconsistency between original flow entries delivered by controllers and the ones preserved by switches. For example, for the convenience of debugging networking, some current OpenFlow switches [5, 6] are left with a listening mode, through which network administrators can connect them from unauthenticated TCP port for manipulating, such as writing rules or reading information. Utilizing such interface, an intentional attacker can eavesdrop certain data streams by artificially adding a mirror port in a switch.

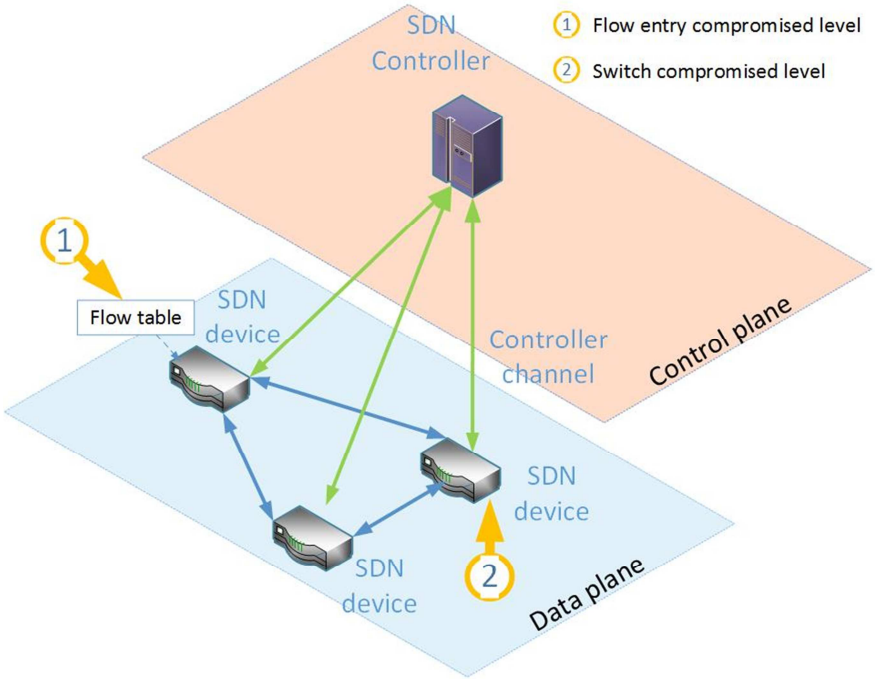

Figure 1. *Eavesdropping in SDN data plane through exploiting switch in two levels.*

Switch compromised level. Directly compromising a switch enables a hacker to monitor the whole traffic flowing through the device, even need not bother to modify the flow entries. Researchers [7] have exploited malware which could hunt out the switch within a targeted network and install a sneaky, second-stage piece of malware on the switch and push data to a command-and-control server. In addition, malicious entity can also pretend regular switches and thus have full visibility into all of the traffic running through the switch.

As for the flow entries compromised level, in traditional network, there are no efficient mechanisms to guarantee the validity of the forwarding rules in devices. However, in SDN, we can utilize its centralized management nature to inspect the integrity of flow entries preserved in switches. As to the switch compromised level, although encryption has demonstrated its reliability in traditional network, in SDN, the scope of protection through encryption is limited. Network encryption mostly is protocol dependent, eg. HTTPS. Since SDN is being designed to open network, it is applied into various areas where customized communication protocols are popular and such protocols haven't adopted corresponding encryption scheme. For example, Data center networking, which is one of the main application domains of SDN, is more frequently using Data Center Interconnect (DCI) protocols, such as Virtual Extensible LAN (VXLAN), Stateless Transport Tunneling (STT), which lack authentication and any form of encryption to secure the packet contents. Thus, we need a protocol oblivious solution.

Such two levels sniffing require different cost from attackers and also reward them distinct benefit. Thus in this paper, we introduce Combat-Sniff, a comprehensive countermeasure, which includes two methods to correspondingly cope with the above two levels situation. Our contributions are as follows:

- We propose a flow entries integrity verification method to deal with flow entries compromised level eavesdropping.
- We propose an innovative protocol oblivious method to prevent data disclosure in switch compromised level eavesdropping.
- We implement the integrated countermeasure, Combat-Sniff, and demonstrate its reliability and performance in experiments.

The structure of the paper is as follows. Firstly, in section 2, we introduce related research works and knowledges. Secondly, we introduce the specifications of Combat-Sniff in Section 3. Later in Section 4, we testify the effectiveness of our methods using experiments. At last, we conclude our work and discuss the subsequent research direction in Section 5.

2. Background

2.1. Related Works

Eavesdropping in SDN can be exerted both within data plane and within the communication channel between controllers and switches. As for data plane eavesdropping, Kevin *et al.* [8] figure out the possibility of adversarial flow tables modifications in OpenFlow network through listening mode of switches. Markku *et al.* [9] analyze such threat in detail by showing us how attackers can utilize flow tables modification as the first-step attacking and then exert eavesdropping. But they didn't give the feasible solutions. Po-Wen *et al.* [10] design a detection mechanism to find compromised OpenFlow switches. Qi *et al.* [11] propose a proactive Random Route Mutation (RRM) technique to

randomly change the routing of multiple flows to defend against eavesdropping. As for communication channel eavesdropping, Kreutz *et al.* [12] propose the vulnerability of the openflow channel, when the attacker captures the flow mod messages, it can modify the messages to add another mirror port. Daniel *et al.* [13] testify such vulnerability through experiments.

Except for eavesdropping, there are also many researches focusing on typical cyber-attacks, which cause serious damages to legacy networks and now developed transformed attacking schemes in SDN. Shin *et al.* [14] figure out how to consume control plane and data plane resource to exert Denial-of-service(Dos) [15] attack in OpenFlow. Later, Shin *et al.* also introduce AVANT-GUARD [16] to relieve such attack by expediting control plane's detection and response ability. Hong *et al.* [17] reveal how to exploit SDN's inherent topology discovery mechanism to poison network visibility and they also construct TopoGuard as an OpenFlow controller extension to secure network topology.

2.2. Background Knowledges

Since our countermeasure involves specific knowledges about OpenFlow protocol and Protocol Oblivious Forwarding [18] technology, we would introduce some related information below.

Flow entries in OpenFlow. In OpenFlow network, every switch preserves multiple flow tables, in which store flow entries installed by controller. Switches would forward data plane packets conforming to flow entries. Each flow entry contains match fields and instructions [19]. The match fields consist of packet headers (eg. TCP_SRC) and if a passing packet matches one of flow entries, i.e. the values of the packet headers equal the values of corresponding match fields of an entry, then the packet would be disposed according to the instructions in the flow entry. A packet not matching any of the flow entries would be sent to controller.

Protocol Oblivious Forwarding. Basing on OpenFlow v1.3, Song [18] proposes Protocol-Oblivious Forwarding. In such forwarding technology, the controller and forwarding elements communicate at a field-offset level rather than protocol semantics level. That is to say, the switch needs not to understand specific packet formats to extract certain search keys, and instead, the controller guides the switch to locate a certain key using a {offset, length} structure. This novel conception allows user to use their own network protocols without any need to go back to the device vendor. This protocol agnostic forwarding device, acts as a facilitating tool to realize our protocol oblivious defense method.

3. Assumptions and Design

3.1. Assumptions and Designing Objectives

Before introduction of our countermeasure, we need to state the following assumptions clearly:

- As for the first-level-based eavesdropping, we assume the sniffer who falsifies the flow entries can't hamper the other OpenFlow functions of that switch.
- As for the second-level-based eavesdropping, we suppose the sniffer who encroaches the whole switch won't hinder the switch from forwarding data packets normally.

For the first assumption, since the first level sniffers' ability is limited to falsifying the flow entries through a feasible interface, it's reasonable for us to think they can't hamper switches' inherent mechanism. For the second assumption, because our focus is networking eavesdropping, we don't consider the other destructive attack from that sniffers. Besides, a switch which doesn't perform normally will easily be detected.

In the design of Combat-Sniff, we plan to achieve the following goals:

- For the first level eavesdropping, we aim to detect the falsifiers efficiently.
- For the second level eavesdropping, we aim to protect communication confidentiality.

3.2. The Sketch of Combat-Sniff

Our countermeasure, Combat-Sniff, includes an active detection method ,called flow entries integrity verification, to realize the sniffing detection goal, and a proactive defense method, called protocol fields randomization, to reach the information confidentiality protection goal.

Method I: Flow Entries Integrity Verification. For a scalable integrity verification solution, the amount of flow entries accesses and transmissions should be minimized, so we utilize a random sampling mechanism to verify part of the flow entries during each round. Besides, we also need to ease the additional burden loaded to switches so as not to influence traffic forwarding. Since the upper controllers can be a centralized cluster of nodes [20] or a physically distributed set of elements [21], we don't need to worry about the storage or the computing capability of the controllers. So we make controller undertake the role of verifier to store the original delivered flow entries and compute the related values.

Design of the method. We adopt a query-reply mechanism, within which the controller periodically sends query messages to switches. On receiving the query messages, switches send the appointed flow entries to controller, and controller verifies their integrity with message digest algorithm (MD5).

The process of verification is in Algorithm 1. Given the total number of flow entries, sampling ratio and total number of switches, in step 1, controller computes the sampling number for each switch according to their share of flow entries. Step 2-10 is the query process. Controller will firstly check the number of flow entries within that switch to inspect if there are maliciously added or deleted flow entries. If not, secondly, it randomly generate the flow entries IDs to be queried and send the message to switches. Step 11-18 is the verification process. Once controller detected inconsistency, it will find the specific flow entry and shut down the suspected ports.

Table 1. *The flow entries integrity verification algorithm.*

Algorithm 1 The flow entries sampling verification algorithm
input: total EntryNum, Sr, SwNum. 1: Computes the sampling number of flow entries for each switch: samp EntryNum; 2: for Sw in switches do 3: Query for the number of flow entries of the switch: entryNum 4: if the value are correct then 5: exert SelectEntry (0, entryNum, sampEntryNum); 6: deliver query messages; 7: else 8: check flow entries item by item in that switch and shut down illegal port; 9: end if 10: end for 11: if Receiving reply from switches then 12: CompareMD5 (OriginalEntry, SampledEntry); 13: if values are not equal then 14: find the specific inconsistent flow entries and shut down illegal port; 15: else 16: pass; 17: end if 18: end if

Method II: Protocol Fields Randomization.

To protect the confidentiality of the data packets which pass through a potential compromised switch, we need to make the switch partially blind. That is to say, the switch should know how to forward the packets, but not know what it is forwarding. As we have mentioned, though traditional communication encryption is a recognized reliable solution, it is protocol dependent.

Inspired by the idea of Address Space Layout Randomization (ASLR) [22], which randomizes the locations of executable segments of a running process to raise the bar for Return-oriented programming(ROP) attack. It occurs to us that we can artificially reorder the locations of protocol fields within a packet scope to enhance the difficulty for sniffers to parse the data packets. Our method has the following features:

- It's protocol oblivious.

It can be applied into any kind of transmission protocol.

- It's content oblivious.

Switch can hardly parse the packet content using regular protocol format knowledges.

Design of the method. We design a specific protocol fields randomization algorithm and a practical transmission scheme to forward the randomized packets.

Protocol fields randomization algorithm. Our randomization scheme includes a reordering and a XOR process. Algorithm 2 represents the specification. There are different levels of reordering according to different data granularities: protocol field level, byte level, bit level. The finer, the safer, but also more expensive for computing. In our defense method, we use byte level reordering.

Table 2. *The protocol fields randomization algorithm.*

Algorithm 2 Protocol fields randomization algorithm
input: A normal data packet with N bytes, within which the header part is M bytes. output: A protocol fields randomized packet 1: Split the single packet into separated N bytes ignoring the header fields meaning. 2: Randomly reorder the M bytes header and distribute them into the whole N bytes scope. 3: Fill in the remaining N − M locations with the original payload of that packet, without changing their relative order. 4: Use a N byte randomly generated secure key to encode the whole packet with XOR

Figure 2 raises an example of the protocol fields reordering process. For simplicity, we only take one protocol field for illustration. In the original TCP packet, the field src IP is located at the offset of 26 bytes of the whole packet ,its length is 4 bytes and value is 10.0.0.2, i.e. 0x0a00 0002 in hexadecimal format. On exerting our protocol fields randomization, we firstly split the integrated field into 4 separate bytes and then reorder them within the whole packet scope. The result is 4 independently bytes, their offsets are k1, k2, k3, k4 respectively and values are 0x02, 0x00, 0x0a, 0x00.

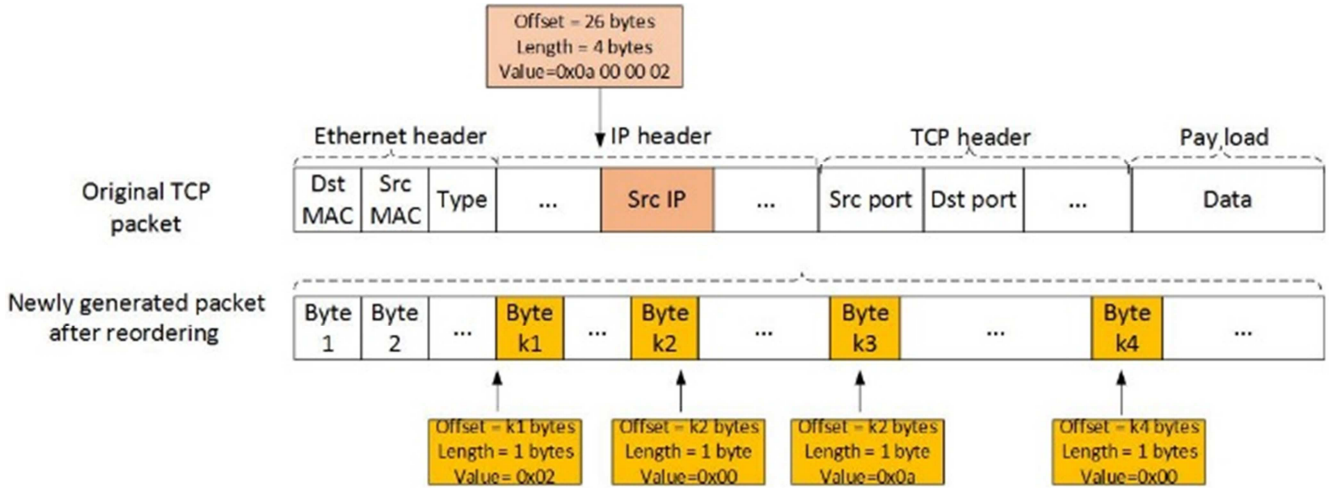

Figure 2. *The reordering mechanism of protocol fields randomization.*

Besides, we also display the actual effect through wireshark in figure 3. For clarity, we only encode the field of source address of IP. The left side is a transformed packet and the right side is an original one. The src IP is 0x0a00 0002. The original field of src IP is reordered and be encoded with XOR. The random key is 0xaaaa aaaa, and the value after XOR should be 0xa8, 0xaa, 0xa0 and 0xaa. We can see that the original field value is distributed into other locations of the packet. And the original location has been substituted by corresponding fields in the packet.

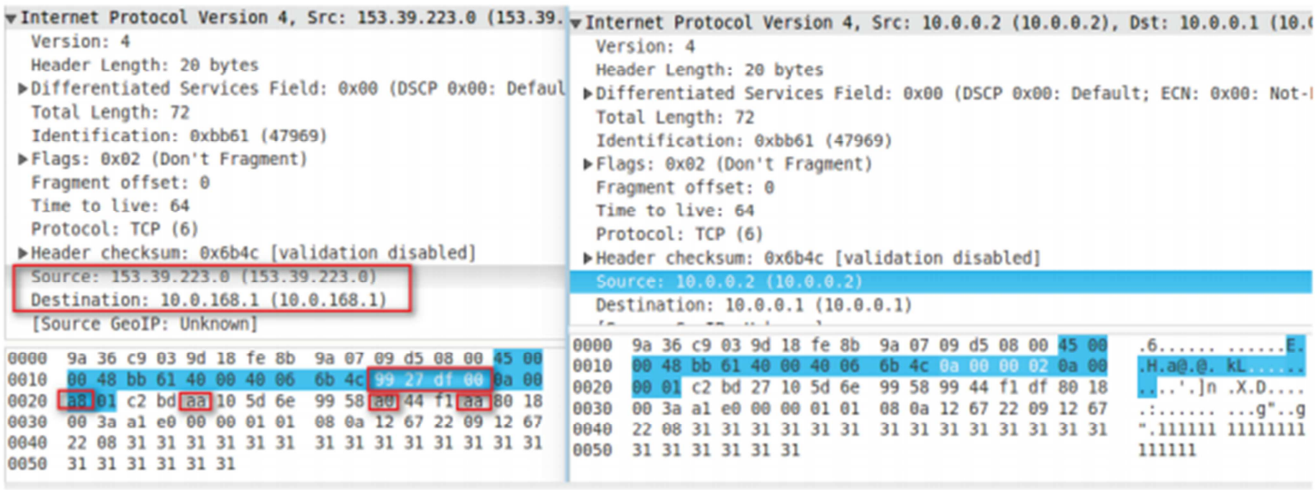

Figure 3. *The comparison of data packet before and after protocol fields randomization.*

The practical transmission scheme. We represent how the protocol fields randomization can be used in practical data transmission in Figure 4.

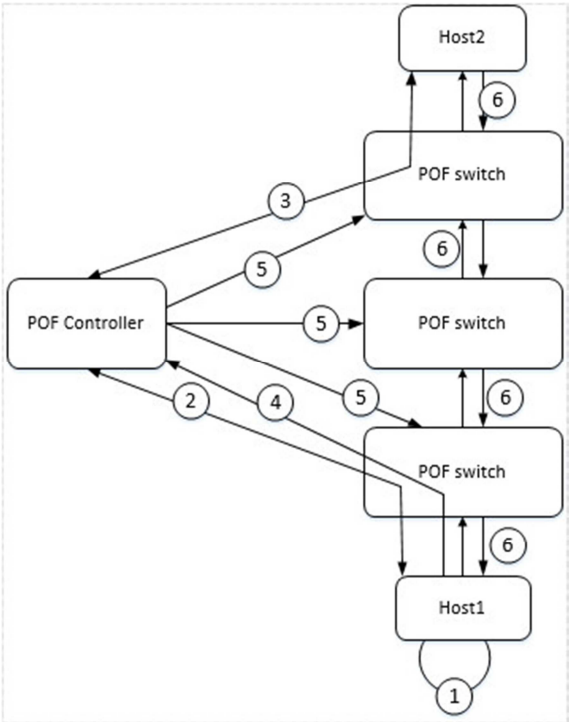

Figure 4. *The practical process of data transmission for protocol fields randomized packets.*

In step 1, host1 will generate its own randomization manners and also appoint some unused fields in a protocol field to mark the corresponding manners. In step 2, host1 shakes hands with controller to inform it the above information and also its communication object host2. Then in step 3, controller would exchange the same information with host2. When the process of handshake succeeds, host1 will communicate with host2 using randomized packets in step 4. And since switch doesn't understand the packet, it would send the packet to the controller. In step 5, controller delivers corresponding flow entries to switches to instruct them to forward the randomized packets between host1 and host2. In step 6, the two hosts can communicate with each other using our protocol fields randomization method.

4. Implementations and Evaluations

Due to our particular function requirement of protocol oblivious forwarding for switches, we use pof controller [23] and pof switch [24] to realize our two methods. And we use pof-mininet plugin to embed pof switch into Mininet to construct a topology. Although the working mechanism of pof is not identical with OpenFlow, but it's based on OpenFlow v1.3 [19] and their distinctions don't hinder our demonstration to our countermeasure.

In realization of the flow entries integrity verification method, we firstly construct a pair of new controller-to-switch OpenFlow messages, FLOW_QUERY and FLOW_REPLY. Secondly, we add a verification module in pof controller and a flow entry reply function in pof switch.

In realization of the protocol fields randomization method, at the host side, we use scapy [25] module of python to encapsulate and parse the randomized packet. At the controller side, we add a packet identification module, it will deliver related flow entries to switches to instruct the packet forwarding.

Our pof controller runs on a physical machine with Intel I5 3.1GHz CPU and 4GB memory. Pof switch-based Mininet

runs on a physical machine with Intel E5-2600 v3 1.6GHZ CPU and 16GB memory.

4.1. Reliability Analysis

(1) Flow Entries Integrity Verification

Assume the attacker tampers E_t flow entries out of total E_n flow entries. And we sample E_s flow entries during every round query. We compute P_x, the probability that at least one of the flow entries selected by sampling matches one of the flow entries tampered by the attacker. Let X be a discrete random variable. It is the number of flow entries selected by sampling that match the flow entries tampered by attacker. So we have:

$$P_x = P\{X \geq 1\} = 1 - P\{x = 0\}$$
$$= 1 - \frac{E_n - E_t}{E_n} \cdot \frac{E_n - 1 - E_t}{E_n - 1} \cdot \frac{E_n - 2 - E_t}{E_n - 2} \cdots \frac{E_n - E_s + 1 - E_t}{E_n - E_s + 1} \quad (1)$$

Since $\frac{E_n - i - E_t}{E_n - i} \gg \frac{E_n - i - 1 - E_t}{E_n - i - 1}$, so $S_r \gg 1 - \left(\frac{E_n - E_t}{E_n}\right)^{E_s}$. We plot figure 5 to express the relationships among sampling ration S_r, detection probability P_x, and total number of flow entries E_n. When tampering ratio i.e. $\frac{E_t}{E_n}$, is a certain probability, say 1%, we can detect the flow entry tampering by sampling a constant amount of flow entries, independently of the total number of the flow entries. For example, if we require a detection probability of 99%, we need only to sample 460 flow entries.

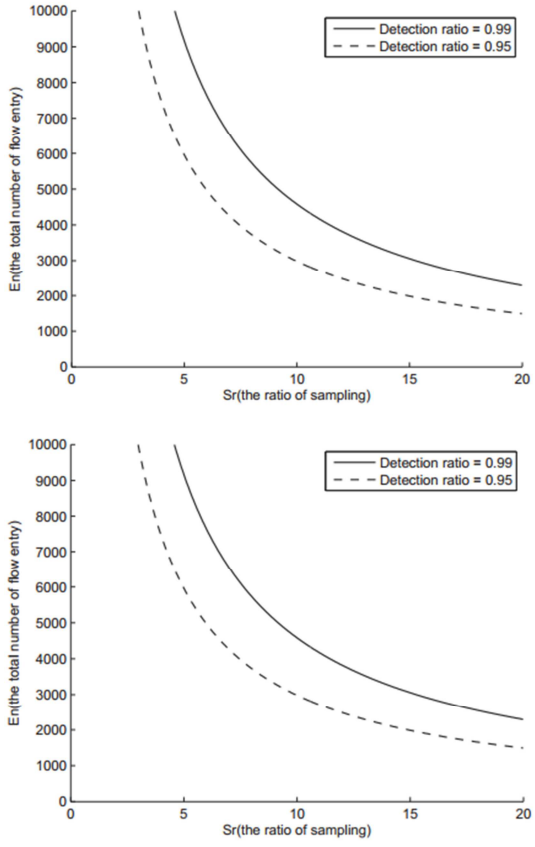

Figure 5. *The probabilistic relationships among sampling ratio, detection probability and total number of flow entries.*

Probabilistic analysis reveals that when the tampering ratio is a constant value, the detection ratio can be pretty high while maintaining a constant number of sampling quantity, independently of the total number of flow entries.

(2) Protocol Fields Randomization

Assume the total length of a data packet is N bytes, and the total number of protocol fields is M. Then the number of reordering schemes, Scheme_N, is computed in equation 2.

$$\text{Scheme}_\text{N} = A_{M+N}^{M}$$
$$= (\text{M} + \text{N}) \cdot (\text{M} + \text{N} - 1) \cdot (\text{M} + \text{N} - 2) \cdots (\text{N} + 1) \quad (2)$$

And after the reordering, we also use a secret key, whose length is 8*(N+M), to encode the packet with XOR. Then we compute P_{parse}, which represents the possibility that an attacker can parse the packet to get the correct value in equation 3:

$$P_{parse} = 1/\{Scheme_N * 2^{8*(M+N)}\} \quad (3)$$

The possibility value demonstrates the difficulty for an attacker to parse the packet and obtain the private information. Of course, we are not specialized in encryption and we just give an instance of our protocol fields randomization method. Professional encryption can be combined to our protocol oblivious method.

4.2. Experiment Effect

(1) Flow Entries Integrity Verification

As the number of flow entries sampled is constant, analyzed in section 4.1, we measure the time needed for sampling different number of flow entries from different number of switches. Table 1 shows that when sampling the same number of flow entries, the more switches we sample, the time consumption is lower. So, we had better collect the sampled flow entries proportionately from as more switches as possible. Because the result data represents that the bottleneck of our method is in performance of a single switch.

Table 3. *The time consumption of flow entries integrity verification in different situations.*

Sampling num / Time(s) / Switch num	1	10	100
1	0.006	1.2	46.056
8	0.005	1.005	25.011
15	0.002	0.038	15.063

(2) Protocol Fields Randomization

We test the performance of our protocol fields randomization method with a file transmission experiment in a 3-switch linear topology. And the time delay comparison with the normal file transmission is in figure 6. When the file size varies from 1KB to 100KB, the transmission delay keeps at about 46%, compared to normal transmission time. The time consumption lies in randomization and de-randomization within communication hosts. And the split

protocol fields also increase the match fields for a switch to search. It is a tradeoff between randomization complexity and data transmission efficiency.

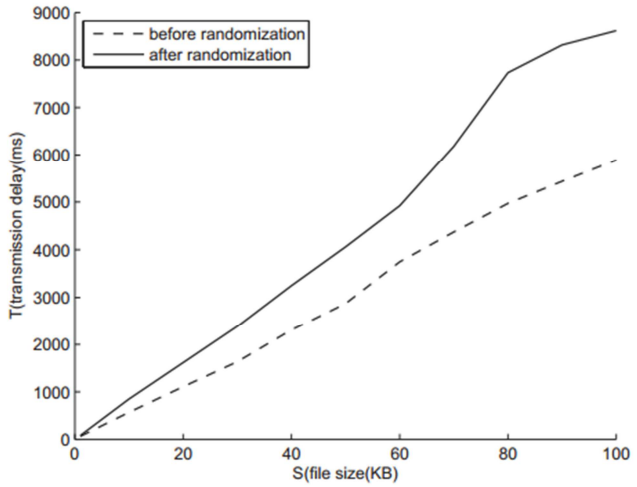

Figure 6. *Comparing transmission delay of different size files.*

5. Conclusion and Future Work

In this paper, we propose Combat-Sniff, a comprehensive countermeasure including two methods to resist data plane eavesdropping in SDN. The detection method can effectively detect the flow entries falsification with a constant number of sampled flow entries. And the defense method enhances a considerable difficulty for attackers to parse the sniffed packets.

In the future work, we plan to research the weight dependent sampling scheme according to the importance of a switch. Besides, we will also try to enhance the process of protocol fields randomization transmission to improve the performance.

Acknowledgements

This material is based upon work supported in part by the Institute of Information Engineering, Chinese Academy of Sciences under Y670021105 and Y5W0011105. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Institute of Information Engineering, Chinese Academy of Sciences.

References

[1] Open Network Foundation. Software-defined networking: the new norm for networks [EB/OL]. [2012-04-13].

[2] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. OpenFlow: enabling innovation in campus networks [J]. ACM.

[3] https://www.owasp.org/index.php/Network_Eavesdropping.

[4] Schultz E E. Assessing and combating the sniffer threat [J]. Local Area Network Handbook, 1999: 85.

[5] Hp switch software - openflow supplement. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c03170243/c03170243.pdf, Feb 2012.

[6] Open vSwitch, 2013. [Online]. Available: http://vswitch.org/

[7] http://www.pcworld.com/article/2957175/sdn-switches-arent-hard-to-compromise-researcher-says.html.

[8] Benton K, Camp L J, Small C. Openflow vulnerability assessment[C]//Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013: 151-152.

[9] M. Antikainen, T. Aura, and M. S¨arel¨a, "Spook in your network: Attacking an SDN with a compromised openflow switch," in Secure IT Systems - 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings, 2014, pp. 229–244.

[10] Chi P W, Kuo C T, Guo J W, et al. How to detect a compromised SDN switch[C]//Network Softwarization (NetSoft), 2015 1st IEEE Conference on. IEEE, 2015: 1-6.

[11] Duan Q, Al-Shaer E, Jafarian H. Efficient random route mutation considering flow and network constraints[C]//Communications and Network Security (CNS), 2013 IEEE Conference on. IEEE, 2013: 260-268.

[12] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. in Proc.2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw., 2013, pp. 55–60

[13] Romão D, van Dijkhuizen N, Konstantaras S, et al. practical security analysis of OpenFlow [J]. 2013.

[14] Shin S, Gu G. Attacking software-defined networks: A first feasibility study[C]//Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013: 165-166.

[15] https://en.wikipedia.org/wiki/Denial-of-service_attack

[16] Shin S, Yegneswaran V, Porras P, et al. Avant-guard: Scalable and vigilant switch flow management in software-defined networks[C]//Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013: 413-424.

[17] Hong S, Xu L, Wang H, et al. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures [C]. NDSS, 2015.

[18] Song H. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane[C]//Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013: 127-132.

[19] Specification, OpenFlow Switch. v1.3.0. (2012).

[20] S. Jain and al., "B4: Experience with a Globally-Deployed Software Defined WAN," in ACM SIGCOMM, 2013.

[21] Berde P, Gerola M, Hart J, et al. ONOS: towards an open, distributed SDN OS[C]//Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014: 1-6.

[22] M. Miller, T. Burrell, and M. Howard. Mitigating software vulnerabilities, July 2011. http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=26788.

[23] http://www.poforwarding.org/pofcontroller-1-1-7-released/

[24] http://www.poforwarding.org/pofswitch-1-3-4-released/

[25] http://www.secdev.org/projects/scapy/doc/usage.html