Source Encryption Scheme in SDN Southbound

Yanlei Wang[1], Shihui Zheng[1](✉), Lize Gu[1], and Yongmei Cai[2]

[1] School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
shihuizh@bupt.edu.cn

[2] School of Computer Science and Engineering, Xinjiang University of Finance and Economics, Urumqi 830000, China

Abstract. In light of the existence of the software defined networking (SDN) southbound communication protocol OpenFlow, and manufacturers' neglect of network security, in this paper, we propose a protection scheme for encryption at the source of the communication data that is based on the Kerberos authentication protocol. This scheme not only completes the identity authentication of and session key assignment for the communication parties on an insecure channel but also employs an efficient AES symmetric encryption algorithm to ensure that messages always exist in the form of ciphertext before they reach the end point and thus obtain end-to-end security protection of communication data. At the end of this paper, we present our experimental results in the form of a forwarding agent. After that, the performance of the Floodlight controller is tested using a CBench testing tool. Our results indicate that the proposed source encryption scheme provides end-to-end encryption of communication data. Although the communication latency increases by approximately 12% when both transport layer security (TLS) and source-encrypted are enabled, the source-encrypted part of the increase is only approximately 4%.

Keywords: SDN · OpenFlow · Source encryption · Kerberos

1 Introduction

Software defined networking (SDN) divides the traditional network architecture into a control plane and a data plane. OpenFlow [1] is the most popular standardized interface between the two planes and has been widely used in academia and industry. Although SDN presents many possibilities for network flexibility and programmability, it also introduces network security threats.

The 1.0 version of the OpenFlow protocol [1] specification contains requirements for TLS usage [2]. However, in subsequent versions of the OpenFlow protocol, [3] "must" is replaced by "should" in their descriptions. Thus, it is difficult to ensure the security of key data, such as flow tables, in southbound communication. In practical applications, few TLS protections exist between controllers and switches. Most vendors ignore southbound communication's security issues and use TCP connections

S. Liu and G. Yang (Eds.): ADHIP 2018, LNICST 279, pp. 560–569, 2019.
https://doi.org/10.1007/978-3-030-19086-6_61

directly. Additionally, the certificate authentication management interface that is used with TLS has not been perfected.

The following table shows the support of Southern OpenFlow TLS for each OpenFlow device vendor [4] (Tables 1 and 2):

Table 1. OpenFlow switch TLS support table

OpenFlow switch	TLS support
HP switch	No
Brocade	Controller port only
Dell	No
NEC	Partial
Indigo	No
Pica8	Only new versions
Open WRT	Yes
Open vSwitch	Yes

Table 2. OpenFlow controller TLS support table

OpenFlow controller	TLS support
Brocade Vyatta controller	Yes
NOX controller	No
POX	No
Beacon	No
Floodlight	No
OpenMuL	No
FlowVisor	No
Big network controller	No
Open vSwitch controller	Yes

Using TCP connections directly ensures that all southbound communication data will be exposed in plaintext on the communication link. This is feasible for secure networks, in which data centers and other physical devices are difficult to access. The number of deployments of the networks is increasing, and in these deployed systems, Without TLS, there will be a serious security risk [5]. Because we cannot guarantee that all communication devices in the communication path between the SDN controller and the switch are secure and reliable, attackers can initiate man-in-the-middle attacks through session hijacking, DNS spoofing, and port mirroring (Fig. 1). They can eavesdrop, intercept, tamper with the communication data, e.g., pose as a controller to send a FLOW_MOD message to the switch, tamper with the flow table in the switch to control the flow of data in the switch, and even tamper with the entire network [6]. However, this kind of attack does not modify the normal transmission from the switch to the controller and is subtle. In [7], this type of attack has been verified in practice as feasible.

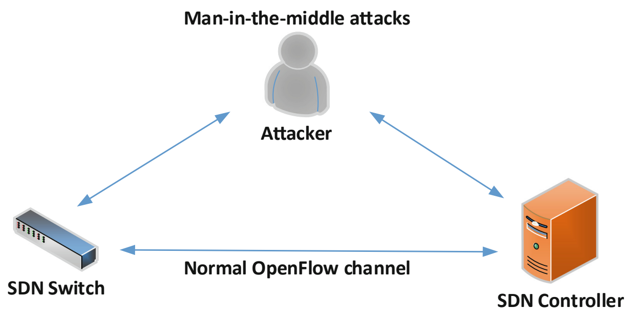

Fig. 1. Man-in-the-middle attacks

The security of the OpenFlow southbound communication channel is not strong. Samociuk et al. [4, 8] proposed using existing security protection schemes to protect the southbound communication. These researchers also analyzed the use of TLS, IPsec, and SSH. The transmission layer and the network layer ensure communication security, but protection schemes focus only on channel security.

In the existing literature, some authors propose security frameworks for the overall assessment and protection of SDN security [9, 10]. Some researchers study the protection of buffer overflow or the denial of service attacks by identifying malicious traffic [11–13]. In some studies [6, 14], a more comprehensive analysis of the SDN architecture and the existing or potential security threats are systematically presented. All [4, 6, 8] authors mention the southbound communication security problem. In addition, all authors discuss the protection of southbound communication security via a channel protection method such as TLS, IPsec or SSH. We believe that the source encryption scheme can completely identity authentication, distribute session keys via insecure communication channel and provide end-to-end communication data security.

The rest of this paper is organized as follows: Sect. 1 introduces the overall architecture of SDN and the OpenFlow southbound communication protocol; Sect. 2 introduces our proposed data source encryption scheme; Sect. 3 contains our program's security assessment and performance evaluation; and Sect. 4 summarizes our work.

2 Source Encryption Scheme in SDN Southbound

2.1 Scheme Profile and Symbol Definitions

SDN architecture is divided into three layers. From top to bottom, they are the application layer, the control plane, and the data plane, as shown in Fig. 2 below. The control plane is responsible for the control logic of the network and provides the calling interface to the application layer. The data plane is responsible for data forwarding and provides the control plane's call interface. Southbound communication refers to the process by which the control plane invokes the protocol interface provided by the data plane to perform network control.

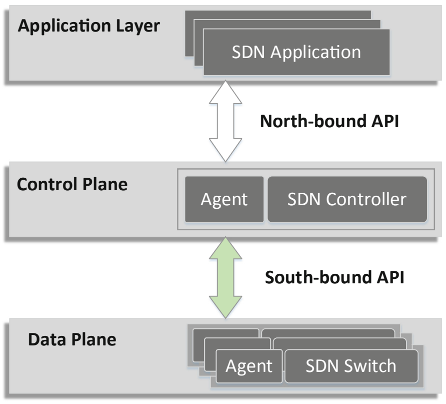

Fig. 2. SDN architecture diagram

The most efficient way to source encrypt the communication data before it enters the communication channel is to internally connect each switch or controller. However, this process causes the system to be highly coupled. In addition, it is not possible for a switch or a controller to be compatible with devices from many manufacturers at the same time. This paper proposes using a data forwarding proxy to implement the source encryption of communication data. As shown in Fig. 1, before the SDN controller and the switch are connected, they are each connected to a local forwarding proxy to perform the encryption and decryption operations on the original data. The abbreviations used to denote the encryption and decryption process are presented below:

$MK_{x,y}$: The key to calculate the HMAC of a message between x and y
$EK_{x,y}$: The symmetric encryption key between x and y
SM_0: The original message from the switch (Plaintext)
SM_{e1}: The source-encrypted message of SM_0 (Ciphertext)
SM_{e2}: The TLS encrypted message of SM_{e1} (Ciphertext)
RM_0: The controller's original message (Plaintext)
RM_{e1}: The source encrypted message of RM_0 (Ciphertext)
RM_{e2}: The TLS encrypted message of RM_{e1} (Ciphertext).

We employ the Kerberos key distribution center (KDC) to perform identity authentication and session key allocation. The abbreviations used to describe the authentication process are presented below:

ID_S, ID_C: The number (unique) of the switch and controller
K_S: The pre-shared key between KDC and the SDN switch
K_C: The pre-shared key between KDC and the SDN controller
T_S: The ticket encrypted by KDC using K_S
T_C: The ticket encrypted by KDC using K_C
$K_{S,C}$: The session key between SDN controller and switch.

2.2 Source Encryption Scheme

A data flow diagram is shown in Fig. 3, in which the virtual devices in the virtual box do not necessarily exist. A device can perform network address translation (NAT), be a firewall, or be an attacker who initiates a man-in-the-middle attack.

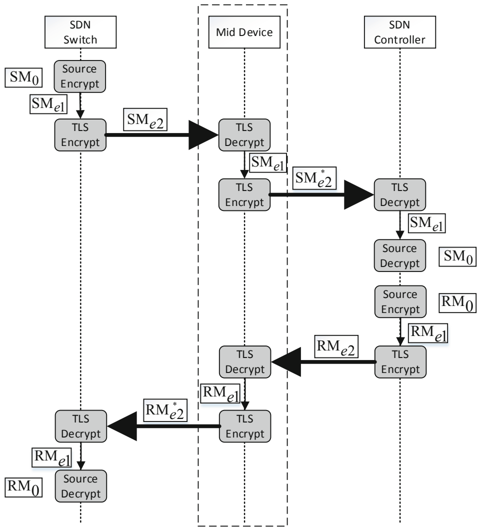

Fig. 3. Source encrypted data flow diagram

SDN switch to SDN controller message encryption and decryption process:

Step 1: The forwarding agent on the switch encrypts the original data from the application layer SM_0 and then calculates the HMAC cipher's authentication code text and adds it to the cipher text SM_{e1}. The calculation is as follows:

$$SM_{e1} = \mathrm{E}\left(EK_{s,c}, SM_0\right) || \mathrm{HMAC}\left(MK_{s,c}, E\left(EK_{s,c}, SM_0\right)\right) \tag{1}$$

Step 2: SM_{e1} is encrypted at the transport layer via TLS, and then the ciphertext SM_{e2} is sent to the intermediate device:

$$SM_{e2} = \mathrm{E}\left(EK_{s,md}, SM_{e1}\right) \tag{2}$$

Step 3: After the intermediate device receives the ciphertext SM_{e2} and uses the TLS key $EK_{s,md}$, which is negotiated by the switch to decrypt it into SM_{e1}, the receiver encrypts SM_{e1} with its TLS key $EK_{c,md}$, which is negotiated with the SDN controller into SM^*_{e2}, and sends it to the SDN control device. The calculations are as follows:

$$SM_{e1} = \mathrm{D}\left(EK_{s,md}, SM_{e2}\right) \tag{3}$$

$$SM_{e2}^{*} = \mathrm{E}\left(EK_{s,md}, SM_{e1}\right) \tag{4}$$

Step 4: After receiving the ciphertext SM_{e2}^{*}, the forwarding agent decrypts it into SM_{e1} using its TLS key. If the message SM_{e1} is verified successfully with the HMAC message authentication code, then it is decrypted into the plain text SM_0 using the source encryption key. If the authentication fails, the message is discarded. The calculations are in Eqs. 5 and 6:

$$SM_{e1} = \mathrm{D}\left(EK_{c,md}, SM_{e2}^{*}\right) \tag{5}$$

$$SM_0 = \mathrm{D}\left(EK_{s,c}, SM_{e1}\right) \tag{6}$$

Encryption and decryption process of Controller to switch message are similar to the above.

2.3 Authentication and Key Distribution

Before each communication between the SDN controller and the switch, the communicating parties must first perform authentication and key distribution to use the assigned key for encrypted communications. The specific authentication distribution process is shown in Fig. 4:

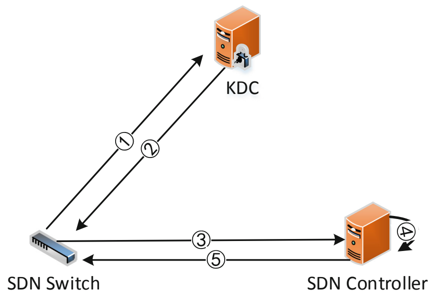

Fig. 4. Authentication process diagram

Step 1: The SDN switch sends a unique number of its own switch and a unique number of the SDN controller to the KDC.
Step 2: After the KDC receives the request, the KDC randomly generates a session key $(K_{S,C})$, and then generates two tickets T_S and T_C to send to the switch. The calculation is as follows:

$$T_S = \mathrm{E}\big(K_S, \big(K_{S,C}, ID_C\big)\big) \tag{7}$$

$$T_C = \mathrm{E}\big(K_C, \big(K_{S,C}, ID_S\big)\big) \tag{8}$$

Step 3: After receiving the two tickets, the SDN switch decrypts T_S with key K_S to obtain the session key $K_{S,C}$ and then uses the session key to generate the authentication factor A with the current time stamp *TS* and the checksum of data *ChS*. A is sent to the SDN controller together with T_C. The authentication factor calculation is as follows:

$$A = \mathrm{E}\big(K_{S,C}, (TS, ChS)\big) \tag{9}$$

Step 4: After receiving the ticket T_C and the authentication factor A, the SDN controller obtains the session key $K_{S,C}$ by decrypting T_C with K_C, and then decrypting A with $K_{S,C}$ to obtain the timestamp *TS* and checksum of data *ChS*. If the timestamp is within five minutes of the current time and if the timestamp is appearing for the first time, then the checksum of data is checked.
Step 5: If certifications pass, then the SDN controller encrypts the timestamp-received *TS* with the session key $K_{S,C}$ and sends it to the switch for the completion of mutual authentication.

3 Analysis and Test

In this section, we demonstrate that the proposed solution is both secure and efficient. We implement the solution in the form of a forwarding agent. For SDN controllers and switches, the source encryption by the forwarding agent is transparent. Controllers and switches are not involved in the encryption and decryption process of communication data, making the protection scheme more flexible and able to adapt to a variety of controllers and switches. Enabling TLS protection for the connection between the security agents at both ends also becomes easier.

3.1 Security Analysis

The source encryption scheme provides message encryption at the transport application layers and provides an identity mutual authentication mechanism. The following is a detailed analysis of the message forwarding process.

As shown in Fig. 3, the messages SM_0 and RM_0, which are sent by the switch or controller, are encrypted before the TLS encryption occurs. During the communication between the switch and the controller, the messages are in the form of ciphertext. Even if an attack on an intermediate device occurs, the intermediate device can only obtain the ciphertexts.

SM_{e1} and RM_{e1} are encrypted at the source. Because only the switch and the controller hold the encryption key $EK_{s,c}$, the intermediate device cannot decrypt the plaintext. Moreover, if the intermediate third-party tampers with the communication message, because the authentication of the message authentication code cannot be performed after the message is received at the receiving end, the receiving end can perceive the security problem on the link.

TLS encrypts data by providing confidentiality protection at the transport layer and providing point-to-point security protection. The source encryption scheme encrypts the message data before the application layer encapsulates the data. The entire communication process is in ciphertext and provides effective end-to-end encryption protection.

During the process of identity authentication and key distribution, if the switch adds a timestamp to the authentication factor that is sent to the controller, the controller must compare the timestamp with the current time and check to see whether it has ever appeared. In this way, the attacker cannot use the replay authentication factor to impersonate the switch for access. The controller also needs to encrypt the time stamp separately and send it back to the switch to complete the mutual authentication.

3.2 Efficiency Test

After adding source encryption protection, we used two desktop computers to test the delays in source encryption, bandwidth throughput, and data packet transmission. The two devices and their parameters are shown in Table 3:

Table 3. Test hardware and software environment

Option	SDN switch	SDN controller
NIC	1000 Mb/s	1000 Mb/s
CPU	Intel i5-4590	Intel i5-4590
Memory	8 GB	8 GB
Kernel version	Linux 4.13.0-38	Linux 4.13.0-38
Operating system	Ubuntu 16.04	Ubuntu 16.04
Software and version	Open vSwitch 2.7.4	Floodlight 1.2

CBench is a tool used to test the performance of OpenFlow controllers. To measure controller performance, CBench can simulate switches to connect controllers, send PACKET-IN messages, and count the number of FLOW-MOD messages to which the controller responds.

To understand the influence of the source encryption scheme on communication latency, we used the CBench test tool to test the performance of the Floodlight controller. We turned on the performance of the controller after the forwarding agent was used as a benchmark, and we separately tested the case of enabling TLS and the source encryption. In the case of both TLS and the enabled source encryption, the controller performance data was compared and analyzed. In each case, we used CBench to simulate different numbers of switches, and we requested the controller at the same time. Each test was repeated 10 times and averaged, as shown in Fig. 5 below:

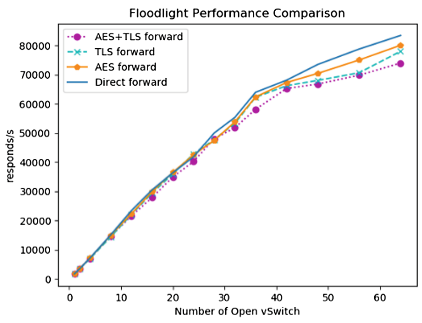

Fig. 5. Controller response performance

In Fig. 5, the horizontal axis indicates the number of switches simulated by CBench, and the vertical axis indicates the number of response messages per second by the controller. When the number of switches is small, the source encryption and the TLS control are separately enabled. The impact of the performance of the device is similar, but when the number of switches is greater than 32, the impact of TLS on the performance of the controller is greater than the source encryption. In general, enabling TLS or source encryption will slightly reduce the performance of the controller. However, if the host resources are sufficient, the controller performance still increases linearly with the number of switches. This does not bring an obvious performance bottleneck to the controller.

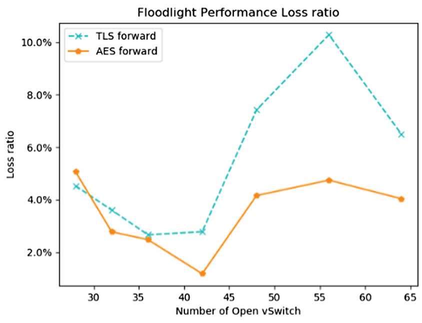

Fig. 6. Loss of controller performance

Figure 6 shows the performance loss of the Floodlight controller when the number of switches ranges from 28 to 64. The controller performance loss when the source encryption alone is enabled is approximately 4%, which is much less than the performance loss when the TLS alone is enabled. Therefore, the source encryption protection scheme proposed in this paper is superior to TLS encryption. When both protection schemes are enabled, the overall performance loss remains below 12%.

4 Conclusion and Future Work

In this paper, we propose a comprehensive and effective source encryption communication scheme, which not only completes the identity authentication and distribution of session keys for both parties of the communication but also provides end-to-end security protection for the communication data. The scheme possesses a high operating efficiency. In addition to studying security protection schemes for communication data on the link, we would like to pursue research into targeted protection schemes for key data in SDN controllers and switches, forming a complete protection system.

Acknowledgement. This work was supported by the National Science Foundation of China (Grant No. 61502048) and the National Science and Technology Major Project (Grant No. 2017YFB0803001).

References

1. Mckeown, N., Anderson, T., Balakrishnan, H., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
2. Consortium, O.F.S.: OpenFlow Switch Specification Version 1.0.0 (2009)
3. OpenFlow switch specifications version 1.4.0. Open Networking Foundation (2013)
4. Benton, K., Camp, L.J., Small, C.: OpenFlow vulnerability assessment. In: ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING, pp. 151–152. ACM (2013)
5. Kobayashi, M., Seetharaman, S., Parulkar, G., et al.: Maturing of OpenFlow and software-defined networking through deployments. Comput. Netw. Int. J. Comput. Telecommun. Netw. **61**(3), 151–175 (2014)
6. Shu, Z., Wan, J., Li, D., et al.: Security in software-defined networking: threats and countermeasures. Mob. Netw. Appl. **21**(5), 1–13 (2016)
7. Yoon, C., Lee, S., Kang, H., et al.: Flow wars: systemizing the attack surface and defenses in software-defined networks. IEEE/ACM Trans. Netw. **25**(6), 3514–3530 (2017)
8. Samociuk, D.: Secure Communication Between OpenFlow Switches and Controllers (2015)
9. Lee, S., Yoon, C., Lee, C., et al.: DELTA: a security assessment framework for software-defined networks. In: Network and Distributed System Security Symposium (2017)
10. Pandya, B., Parmar, S., Saquib, Z., et al.: Framework for securing SDN southbound communication. In: 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), pp. 1–5. IEEE (2017)
11. Ambrosin, M., Conti, M., Gaspari, F.D., et al.: LineSwitch: tackling control plane saturation attacks in software-defined networking. IEEE/ACM Trans. Netw. **25**(2), 1206–1219 (2017)
12. Atli, A.V., Uluderya, M.S., Tatlicioglu, S., et al.: Protecting SDN controller with per-flow buffering inside OpenFlow switches. In: Black Sea Conference on Communications and NETWORKING. IEEE (2018)
13. Deng, S., Gao, X., Lu, Z., et al.: Packet injection attack and its defense in software-defined networks. IEEE Trans. Inf. Forensics Secur. **13**(3), 695–705 (2017)
14. Rawat, D.B., Reddy, S.R.: Software defined networking architecture, security and energy efficiency: a survey. IEEE Commun. Surv. Tutorials **19**(1), 325–346 (2017)