Detection of Incongruent Firewall Rules and Flow Rules in SDN

Nandita Pallavi, A.S. Anisha and V. Leena

Abstract The networking is the backbone that supports the vast area of Information Technology. SDN is the new road that takes the conventional networking to greater heights. SDN is going to aid all future innovations and developments in the field of networking. SDN stands for Software Defined Networking, this separates the network into two planes namely data plane and control plane. A data plane is the abstraction of all the hardware side of the network and the control plane is the central unit that acts like a brain controlling the entire network. This dual architecture thus helps to maintain a network that is centralized, highly scalable, flexible etc. The programmability of the network opens the window of scope for greater innovations and developments. SDN can gracefully accommodate technology shifts. At the same time SDN posses certain security issues that need to be addressed. As a widely flourishing and developing networking method, these security issues need to be tackled. In this paper we are trying to address the security issue of rewriting flow entries in switches. We propose an algorithm for the detection of incongruence between firewall rules and flow rules and thus we overcome the threat caused by modification of flow entries. The proposed system is for Open Flow based Firewalls. The system is intended to boost the security capabilities of SDN, thereby minimizing some of the security challenges in SDN.

Keywords SDN · Firewall · Open flow · HSA · Firewall rules · Flow rules · Duo lock

N. Pallavi (✉) · A.S. Anisha · V. Leena
Department of CS & IT, Amrita School of Arts and Sciences Kochi,
Amrita Vishwa Vidyapeetham, Kochi, India
e-mail: pallavinandita@gmail.com

A.S. Anisha
e-mail: anishaambili007@gmail.com

V. Leena
e-mail: vleena@gmail.com

S.S. Dash et al. (eds.), *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, Advances in Intelligent Systems and Computing 517,
DOI 10.1007/978-981-10-3174-8_2

1 Introduction

Software Defined Networking is an emerging architecture that is cost-effective, dynamic, manageable, and adaptable [1]. All these features make it ideal for the high-bandwidth, dynamic nature of today's applications [2]. Using this technology a third party can introduce a new service or customize network behavior by writing simple software. Coming to the security aspect in SDN [3, 4], though some level of security [5] is provided by the SDN switches, it does not provide enough protection from all the things that can go wrong. This is where the importance of a firewall in SDN is reinstituted. The firewall filters all the incoming and outgoing traffic in the network based on prioritized firewall rules [6]. A firewall constitutes of a collection of rules that either allow or deny traffic. The flow tables are present in the switches and the entries in flow tables mark the path the traffic can traverse. There may be contradiction between the flow rules and flow table entries [7]. This contradiction is the violation that we are handling in this paper. The packet entering the switch passes through flow tables. A flow is a sequence of packets that matches a specific entry in a flow table.

2 Related Work

Hu et al. [1] has presented a comprehensive framework, FLOWGUARD, has proposed a firewall policy violation detection and resolution mechanism in dynamic OpenFlow networks. Jarschel [8] establish SDN as a widely adopted technology beyond laboratories and suggest that insular deployments requires a compass to navigate the multitude of ideas and concepts that make up SDN today [8]. Contribution represents an important step towards such an instrument. It gives a thorough definition of SDN and its interfaces as well as a list of its key attributes. Mininet [9] an instant virtual platform http://mininet.org/ helps to understand the implementation level details of SDN. Hu et al. [7] proposed that the source and destination addresses of firewall rules and flow entries are first represented by binary vector. Then, conflicts between firewall rules and flow rules are checked through comparing the shifted flow space and deny firewall authorization space. PeymanKazemian et al. [10] has developed a general and protocol-agnostic framework, called *Header Space Analysis* (HSA). Their formalism allows us to statically check network specifications and configurations to identify an important class of failures such as *Reachability* Failures, *Forwarding Loops* and *Traffic Isolation and Leakage* problems. Wang et al. [11] introduce a systematic solution for conflict detection and resolution in OpenFlow-based firewalls through checking flow space and firewall authorization space. This approach can check the conflicts between the firewall rules and flow policies based on the entire flow paths within an OpenFlow network. PeymanKazemian et al. [12] introduces a real time policy checking tool called NetPlumber.

3 Methodology

3.1 *Algorithm 1: Matrix Mapper*

Step 1: Convert the firewall rules into a matrix format.
Step 2: Convert the flow table rule into a matrix format.
Step 3: Create a transitive adjacency matrix based on the flow table matrix.
Step 4: By comparing firewall table and flow table matrices, derive a resultant matrix and also create a port matrix.

Based on the topology given in Fig. 1, matrices are designed here. The firewall matrix consists of values 0, 1 and 2 and it is denoted by the symbol FW. Row and column denotes the source and destination respectively. Each value in the cell of the matrix corresponds to the state of traffic between the source and destination. The state of traffic implies whether there is traffic between the corresponding source host and destination host [10, 12]. The value 0 means traffic denial, 1 means traffic is allowed, and 2 denotes partial allow. The firewall rules are analyzed from least priority to high priority to set values in the matrix. If allow or deny between two hosts is not specified then it is assumed to be allow. Table 2 is an FW matrix designed from the firewall table given in Table 1.

The Flow table matrix contains two values 0 and 1 where 0 corresponds to absence of flow and 1 corresponds to the presence of flow. It is represented by the symbol FT.

From matrix FT, transitive adjacency matrix TA is constructed. The significance and procedure of creating matrix TA from a given matrix is explained in Sect. 3.3. The corresponding TA matrix of FT shown in Table 3 is given in Table 4.

The matrix FW and TA are compared. From these two matrices resultant matrix *R* is built. Whenever a new rule is inserted or an existing rule is dropped in FW or an existing rule is modified, the *R* matrix is built. For values of matrices with values 1 or 0, an XNOR operation is performed and the result is recorded in matrix *R*. The value 0 in *R* means violation. That is the case when an action is specified in firewall and a contradicting action is specified in flow table (Table 5).

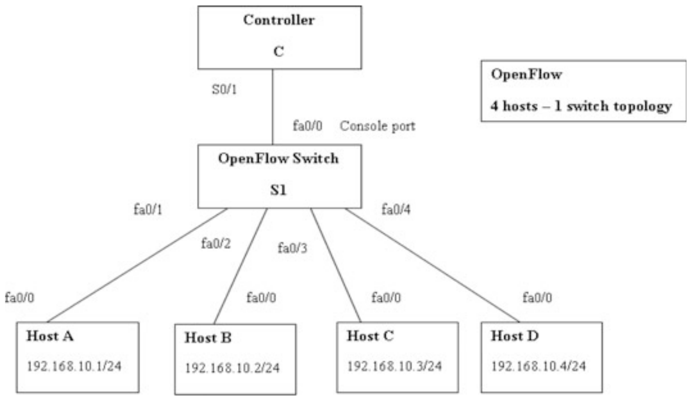


Fig. 1 SDN test topology

Table 1 Firewall table

Order	Protocol	SrcIP	*	DestIP	Dest port	Action
1	tcp	192.168.10.1	*	192.168.10.2	*	Allow
2	tcp	192.168.10.2	*	192.168.10.4	22, 25	Deny
3	tcp	192.168.10.3	*	192.168.10.2	*	Deny
4	tcp	192.168.10.4	*	192.168.10.2	*	Deny
5	tcp	192.168.10.4	*	192.168.10.3	*	Allow
6	tcp	192.168.10.3	*	192.168.10.4	*	Allow
7	tcp	192.168.10.2	*	192.168.10.1	22, 23	Allow
8	tcp	192.168.10.3	*	192.168.10.1	*	Deny
9	tcp	192.168.10.2	*	192.168.10.3	*	Deny
10	tcp	192.168.10.1	*	192.168.10.4	*	Deny

Table 2 Firewall matrix: FW

	A	B	C	D
A	1	1	1	0
B	2	1	0	2
C	0	0	1	1
D	1	0	1	1

Table 3 Flow table matrix

	A	B	C	D
A	1	1	1	0
B	0	1	0	1
C	0	1	1	0
D	0	1	0	1

Table 4 Transitive adjacency matrix

	A	B	C	D
A	1	1	1	1
B	0	1	0	1
C	0	1	1	1
D	0	1	0	1

Table 5 Table of XNOR operations

Inputs		Output
A	B	$Y = \overline{A \oplus B} = AB + \overline{A}\,\overline{B}$
0	0	1
0	1	0
1	0	0
1	1	1

If the matrix FW has a value 2 and TA has value 1, value 2 is recorded in *R*. When FW has a value 2 and TA has value 0, it is a violation and therefore value 0 is recorded in *R* (Table 6).

Table 6 Resultant matrix *R*

	A	B	C	D
A	1	1	1	0
B	0	1	1	2
C	1	0	1	1
D	0	0	0	1

Table 7 Port matrix of source B

	20	21	22	23	50	53
A	0	0	1	1	0	0

The matrix *R* helps us to detect all the incongruencies in cases other than those with partial allow (value 2), for the remaining entries (with value 2) there is a port specification associated with it. Therefore a next level of matrix needs to be constructed.

If $R_{ij} = 2$ in *R*, get the corresponding hosts. For all these hosts, create a port set for each host. With each cell with value 2, create a destination-port matrix for the corresponding source. The row contains the destination host names and column contains the partially allowed ports. An example for port matrix is shown in Table 7.

This helps to record the violation between the firewall rules and flow rules in SDN. The prioritizing errors can be detected using this strategy.

(1) A value 2 in FW matrix cannot be overwritten by 1 or 0, that is a priority violation.
(2) If specific ports of host are allowed or denied, the rest of the ports are filled with the negated value of either allow or deny.

3.2 Description of Duo Lock Algorithm

As the name suggests the algorithm is governed by two locks. The outer lock: lock_1 and the inner lock: lock_2. Whenever there is a modification in a flow entry or a firewall rule is added, deleted or modified, the control is passed through this lock. The entry is updated only when the lock is opened. If the lock is closed a violation is detected. The flow entry is updated when a mandatory overwriting is tried or changes need to be brought in the flow table based on firewall rule addition, modification or deletion. Lock_1 helps to detect and handle complete violations. In case of partial allow or deny rules in firewall, the second lock: lock_2 is used. If port wise violation does not happen then the lock is opened and flow entry is updated, otherwise if the port is closed, flow entry does not get updated and violation is reported.

In order to enable the proper execution of the concept, we introduce a transitive adjacency matrix in the locks. If the hosts A and C are denied to interact, the intruder must not be allowed to bypass this rule by an indirect interaction. Example is an indirect interaction from A to C by A to B to C. Lock_1 operates using transitive adjacency matrix and Lock_2 operates using port matrix.

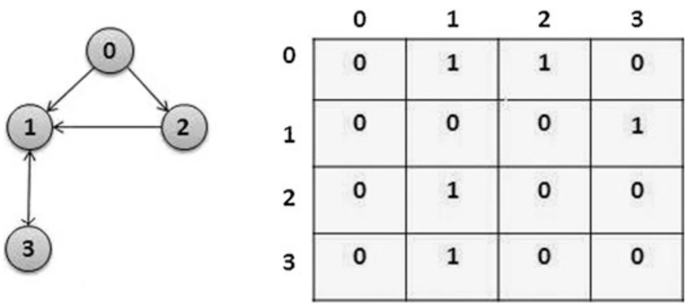
3.3 Transitive Adjacency Matrix (TA)

A transitive adjacency matrix is the matrix that points whether there is a direct edge or indirect edge in a matrix. If edge (a, b) and (b, c) exists then another edge (a, c) is assumed to be existing. The transitive adjacency matrix is based on transitive law and adjacency matrix. The transitive law in mathematics and logic states that "If $a\mathbf{R}b$ and $b\mathbf{R}c$, then $a\mathbf{R}c$," where "**R**" is a particular relation (e.g., "… is equal to …"), a, b, c are variables (terms that may be replaced with objects) and the result of replacing a, b, and c with objects is always a true sentence (Fig. 2).

A transitive adjacency matrix is a matrix with all indirect connections also marked as an edge. This matrix is XNORed with FW matrix, if 1 then there is violation and the lock_1 is locked (Table 8).

For converting an adjacency matrix to transitive adjacency matrix a set of operations are proposed here.

```
    For each row, $r$ in matrix with value 1{
    Find value of column with value 1 and assign column value to $k$
For each row of value k, retrieve column with value 1 and assign to l.
    Assign TA[$r$][l] to 1.
```



	0	1	2	3
0	0	1	1	0
1	0	0	0	1
2	0	1	0	0
3	0	1	0	0

Fig. 2 Adjacency matrix representation of a directed graph

Table 8 Transitive adjacency matrix of the adjacency matrix given in Fig. 2

	0	1	2	3
0	0	1	1	1
1	0	1	0	1
2	0	1	0	1
3	0	1	0	1

3.4 Algorithm 2: Duo Lock

```
Lock1
//a prevention is better than cure approach
Matrix R (Resultant matrix of the comparison between FW and TA) checked,
If(value is 1)
        //no violation at all
        Exit
If(value is 0)
        //lock closed//violation detection
        Set lock _1 to 0

        return closed
Else
      //Lock opened
        Set lock_1 to 1

        return open

Lock 2
//cure approach
Matrix model Port matrix checked, on value 1 lock opened
Based on source destination variables corresponding port matrix is analyzed
If port deny
      //close lock_2
      Set lock_2 to 0
      return closed
Else
      //lock opened
      Set lock_2 to 1
        return open

Duo Lock:

Lock1{
      If (closed){
              Violation detection
      }
      Else {
              Flow entry updation

                      lock2{
                      If (closed){
                              Violation detection
                              Exit
                      }
                      Else{
                              //no violation
                              Flow entry updated if needed
                              Exit
                      }
              }
      }
}
```

The proposed methodology was experimented in mininet [9] using pox controller [13, 14] based on the topology given in Fig. 1. From the experiment it was clear that rules can be efficiently represented, stored and manipulated effectively using matrices. In terms of memory access also matrices are highly efficient here. As 2 check points are introduced no conflict can be bypassed. So the proposed procedure effectively detects the conflicts in a faster way. Therefore flow tables can be updated reflecting the variation proficiently.

4 Conclusion

In this paper we present a method of detection of incongruent firewall rules and flow rules in SDN. Our methods are based on the matrix model. Firewall matrix, flow table matrix, port matrix and transitive adjacency matrix is the matrix models used in this paper. Using these matrix models we have created two algorithms to detect the violations between flow table entries and flow rules. These algorithms provide an efficient way to detect violations and prevent or cure their occurrence. A method to effectively prioritize the firewall rules depending on modification, insertion or deletion of firewall rules have also been proposed in this paper. The matrix model of transitive adjacency matrix is a matrix that has been formulated according to the needs of our method, and for this we have clubbed the transitive law and adjacency matrix.

References

1. Hongxin Hu, Wonkyu Han, Gail-JoonAhn and Ziming Zhao “FLOWGUARD: Building Robust Firewalls for Software-Defined Networks” Clemson University Arizona State University.
2. Wolfgang Braun and Michael Menth “Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices” Department of Computer Science, University of Tuebingen, Sand 13, Tuebingen 72076, Germany.
3. Phillip Porras, Steven Cheung, MartinFong, Keith Skinner, and VinodYegneswaran “Securing the Software-Defined Network Control Layer”.
4. Seunghyeon Lee, Chanhee Lee, Hyeonseong Jo, Jinwoo Kim, Seungsoo Lee, Jaehyun Nam, Taejune Park, Changhoon Yoon, Yeonkeun Kim, Heedo Kang, and Seungwon Shin “A Playground for Software-Defined Networking Security” GSIS, School of Computing, KAIST.
5. Jérôme François, LautaroDolberg, Olivier Festor, Thomas EngelSnT “Network Security through Software Defined Networking: a Survey” - University of Luxembourg.
6. Michelle Suh, SaeHyong Park, Byungjoon Lee, Sunhee Yang “Building Firewall over the Software-Defined Network Controller” SDN Research Section, ETRI (Electronics and Telecommunications Research Institute), Korea.
7. Hongxin Hu, Wonkyu Han, Gail-JoonAhn, and ZimingZhao “Towards a reliable SDN firewall” Clemson University Arizona State University.
8. Michael Jarschel, Thomas Zinner, Tobias Hobfeld, Phuoc Tran-Gia. “Interfaces, attributes and use cases—a compass for SDN”.

9. Mininet, an instant virtual platform http://mininet.org/.
10. PeymanKazemian, Nick McKeown, George Varghese “Header Space Analysis: Static Checking For Networks” Stanford University, UCSD and Yahoo! Research.
11. Juan Wang, Yang Wang, Hongxin Hu, Qingxin Sun, He Shi, and LangjieZeng. “Towards a Security-Enhanced Firewall Application for Openflow Network”.
12. PeymanKazemian, Michael Chang, HongyiZeng, George Varghese, Nick McKeown, Scott Whyte “Real Time Network Policy Checking using Header Space Analysis”.
13. Pooja, Manu Sood “SDN and Mininet: Some Basic Concepts” Department of Computer Science, Himachal Pradesh University, Shimla.
14. Sukhveer Kaur1, Japinder Singh2 and Navtej Singh Ghumman “Network Programmability Using POX Controller” 3 1,2,3 Department of Computer Science and Engineering, SBS State Technical Campus, Ferozepur, India.