Towards Secured Firewalls for Software Defined Networks

D N Kartheek
Department of CSSE, Sree Vidyanikethan Engineering College, Tirupathi, India

Abstract—Software-Defined Networking (SDN) offers programmers network-wide visibility and direct control over the underlying switches from a logically-centralized controller. SDN provides a promising way for the future development of Internet. SDN, however, also has some new security challenges. A critical challenge among them is how to build a reliable firewall application for SDN. Due to the stateless property of SDN firewall based on OpenFlow, it lacks audit and tracking mechanisms, also the existing firewall applications in SDN can be easily bypassed by rewriting the flow entries in switches. Focusing at this threat, we introduced a novel solution for conflict detection and resolution in SDN firewall based on OpenFlow-based firewalls through checking flow space and firewall authorization space. Unlike FortNOX, our approach can check the conflicts between the firewall rules and flow policies based on the entire flow paths within an OpenFlow network. Finally we added intra-table dependency checking for flow tables and firewall rules.

Keywords— Software Defined Networks, Firewalls, Header Space Analysis.

I. INTRODUCTION

Operating and maintaining a computer network is an arduous task. To express the required high-level network policies, network operators need to configure each individual network device separately from a heterogeneous collection of switches, routers, middleboxes, etc. using vendor specific and low-level commands. In addition to configuration complexity, networks are dynamic and operators have little or no mechanisms to automatically respond to network events. It is therefore difficult to enforce the required policies in such a continually changing environment. With the separation of control plane from data plane that lays the ground to the Software Defined Networking paradigm, network switches become simple forwarding devices and control logic is implemented in a logically centralized controller.

Software Defined Networking (SDN) is a novel network framework introduced at Stanford University. This enables programmers to control and define the networks by software programming, which makes it be regarded as improvement in the networking field. As the core technology of SDN, OpenFlow (OF) [1] is a novel network transfer model which separates the functionalities of the network controlling and flows. In this model users can control the operation of the packets in the networks by inserting flow entries into the switches. In a traditional network data plane and control plane are implemented in switches and routers, whereas SDN decouples those two planes. In an SDN, the control plane using a new protocol called OpenFlow protocol controls the flow tables in the switches. In this way, the control plane realizes the centralized control to the whole network. A controller will compute the shortest flow paths for specific work and control the forwarding behaviors made by the switches. The controller could be a device, a virtual computer, or a physical server [2].

SDN has two vital characteristics. First, an SDN separates the control plane (which decides how to handle the traffic) from the data plane (which forwards the traffic according to decisions that the control plane makes). Second, an SDN consolidates the control plane, so that a single software control program controls multiple data plane elements. The SDN control plane exercises direct control over the state in the network’s data plane elements (i.e., routers, switches, and other middleboxes) via well defined Application Programming Interface (API). OpenFlow is a prominent example of such API. An OpenFlow switch has one or more tables of packet-handling rules. Each rule matches a subset of traffic and performs certain actions on the traffic that matches a rule; actions include dropping, forwarding, or flooding. Depending on the rules installed by a controller application, an OpenFlow switch can behave like a router, switch, firewall, network address translator, or something in between [17].

Although SDN introduces many advantages in the development of networks, it also brings forth some new security challenges. One such challenge is how to build a secure firewall application for SDN networks. A major limitation of OpenFlow is that it is stateless. For instance, if a host or a network device sends a flow to the network, only the first packet of the flow will be checked by the controller while the consequent packets will be directly forwarded by the switches without any exploration. Also there is no audit or tracking mechanism set towards flows in the controller. Therefore, the existing firewall application for SDN could be easily bypassed by inserting the flow entries with rewriting operations deliberately [14].

To address such a threat, a systematic approach is introduced for conflict detection and resolution in SDN firewall via checking flow space and firewall authorization space. First search the flow paths in the entire network and check them against all firewall deny rules to find out whether the flow paths conflict with the firewall rules. Then, present different conflict resolution strategies according to different operations.
in the firewall rules or the flow tables. Considering that the address space of a flow path may be different from the address space of the conflicting firewall rules, introduced a method that is to insert specific blocking firewall rules. They justifies the traffic coming IN (Ingress) and OUT (Egress). They are not applied to physical ports. They just apply to where the traffic is coming from and where it’s going to. It’s all virtual, which is why they are called ingress and egress instead of ‘incoming’ and ‘outgoing’ ports) of the flow path. Through this method, the firewall application could block the packages that are in conflict with the firewall rules without disrupting other normal packages. By creating and maintaining a shifted flow graph, a tracking mechanism for flows can be established and therefore solve the bypass problem fundamentally.

II. RELATED WORK

With the quick developments of SDN techniques, the security issues in SDN have attracted more attention recently.

Son et al., [1] introduced FLOVER, a model checking system, which verifies that the aggregate of flow policies instantiated within an OpenFlow network does not violate the network security policy. Their system detects faults leading to invalid and invisible routes, but it doesn’t consider firewall policies.

R.Sherwood et al., [2][3] propose FlowVisor which enables secure network operations by segmentation, or slicing, network control into independent virtual machines. Each network domain is governed by a self-consistent OF application, which is architectured to not interfere with OF applications that govern other network slices. In this sense, OpenFlow security is casted as a non-interference property. However, even within a given network slice the problem remains that a network operator may still want to instantiate network security constraints that must be enforced within the slice.

A.Liu et al., [4] proposed firewall verification toll which takes input a firewall policy and a given property, then outputs whether the policy satisfies the property. In this, Liu designed and implemented a verification algorithm using decision diagrams, and tested it on both real-life firewall policies and synthetic firewall policies of large sizes.

A.Liu et al., [5] done some work on modelling firewall security policies. However, these studies do not address the dynamic nature of flow rules in software defined networks.

E. Al-Shaer et al., [6] done some work related to FlowChecker, which encodes OF flow tables into Binary Decision Diagram (BDD) and uses model checking to verify security properties and verifies which proposes to slice the OF network into equivalence classes to efficiently check for invariant property violations [7]. However, these systems do not explicitly address intermediate actions such as set and goto commands.

M. Canini et al., [8] proposed NICE which uses symbolic execution to verify conformance of OF applications. However, such path exploration approaches do not scale well for large applications.

N. Mckeown et al., [9] introduced a concept of OpenFlow switches and used in different applications such as NOX (Operating system for networks) and FlowVisor (A network virtualized layer). The work done on OpenFlow switches did not address the problems of conflict analysis and model verification; instead, it showed the basic architecture of OpenFlow model and how it can be used to provide logical separated networks on the physical network.

R.Sherwood et al., [10] introduced network virtualization layer, where by a production network is sliced to multiple virtual networks that run multiple experiments simultaneously, each with their own forwarding decision.

M. Casado et al., [11][12] proposed a new architecture for securing enterprise networks. The SANE [11] protection layer proposes a fork-lift (clean-slate) approach to upgrading enterprise network security that introduces a centralized server, i.e., domain controller, to authenticate all elements in the network and grant access to services in the form of capabilities that are enforced at each switch. Ethane [12] is a more practical and backwards-compatible instantiation of SANE that requires no modification to end hosts. Ethane switches reside alongside traditional network switches and communicate with the centralized controller that implements policy. Both studies may be considered as catalysts for the emergence of OpenFlow and software-defined networking

Wen et al., [13] proposed PermOF, a fine-grained permission system, as the first line of defense to apply minimum privilege on applications. They summarized a set of 18 permissions to be enforced at the API entry of the controller.

Porras et al., [14] proposed FortNOX, which uses single IP address to identify potential bypass violations and provides a conflict detection and resolution approach. However, it detects conflicts based on pair wise comparisons, while ignores rule dependency in both flow tables and firewall policies. Also, it could not build a globe view of network state.

Kazemian et al., [15] has done some work which could enable dynamic reachability checking and build a globe view of network state by real-time network verification. Even though this approach can be applied for policy conflict detection, they could not provide effective mechanisms for policy conflict resolution.

Nate Foster et al., [16] introduced Frenetic Language which offers programmers a collection of powerful abstractions for writing controller programs for software defined networks. A compiler and run time systems implements these abstractions and ensures that programs written against them execute efficiently. The work in this paper focuses on the three stages of managing a network – monitoring network state, computing new policies and reconfiguring the network.
SDN, as a new network paradigm, was just introduced a couple of years ago. Because it allows network applications to operate with switches in the networks directly, it faces a variety of security challenges. The following are the identified research gaps:

- In Existing approaches [1] the firewall policies doesn’t consider the detection of faults leading to invalid and invisible routes.
- Current OpenFlow systems are stateless and do not address the dynamic nature of flow rules in software defined networks [4][5].
- The conflict among the various applications is not considered in existing SDN controllers [15].
- In existing architecture [14], the firewalls or other security applications can be easily bypassed by adding deliberated flow tables.
- The existing approaches [14] do not consider intra-table dependency among the firewall rules and flow entries.
- There is no audit or tracking mechanisms set towards flows in the controller in existing system.
- Existing systems [8] do not scale well for large applications.

In this regard, the above challenges have been identified as objectives for the proposed work.

IV. IMPORTANCE OF PROPOSED WORK

Software Defined Networking (SDN), offers programmer network-wide visibility and control over the underlying switches from a logically-centralized controller, not only has a huge impact on the development of computer networks, but also provides a promising way for the future development of internet. There are many security challenges that come into picture. One such challenge is how to build a secure firewall application for SDN. Due to the stateless property of SDN firewall based on OpenFlow and the deficiency of tracking and audit mechanisms, the existing SDN firewalls can be easily bypassed by rewriting the flow entries in switches. Focusing on this threat, a systematic approach for conflict detection and resolution in OpenFlow-based firewalls through checking flow space and firewall authorization space is introduced. Unlike FortNOX [14], the proposed approach can check the conflicts between the firewall rules and flow policies based on the entire flow paths within an OpenFlow network. Also addition of intra-table dependency checking for flow tables and firewall rules is to be done. Finally, a proof-of-concept implementation of the proposed approach is discussed and experimental results will demonstrate that the approach can effectively avoid the bypass threats in real OpenFlow networks.

V. METHODOLOGY

A. Header Space Analysis:

The proposed conflict detection and resolution algorithm is based on Header Space Analysis. A uniform and protocol-agnostic model of the network using a geometric model of packet processing is provided by HSA. A header is defined as a point in space \( \{0,1\}^L \) which is called header space (L is the length(in bits) of the packet). Network boxes are modelled using a Switch Transfer Function T, which transfers received header h in the input port to a set of packet headers on one or more output ports:

\[ T: (h,p) \rightarrow \{(h1,p1),(h2,p2),.......\} \]

B. Shifted Flow Space and Authorization Space:

In order to check whether the firewall rules conflict with flow tables in OpenFlow switches, all the packets should be tracked and calculation of all the destinations which the packets can reach and the header space at each destination is to be done. The source address and destination address in the header space of every flow paths with the address space derived from the firewall policy is to be compared. If they have intersection, the flow rules are considered in conflict with the firewall policy.

Generally, the firewall rule consists of 5 fields: source address, source port, destination address, destination port, and protocol. The ingress header space of a flow path consists of three fields: source address, source port, and protocol. The egress header space of the flow path consists of two fields: destination address and destination port. Through the ingress and egress space which constructs a tracked space of a flow path, the source and the destination of a traffic flow path is to be figured out. All flow paths form a graph, which is called the netplumbing graph [15].

The proposed conflict resolution approach just considers the flow paths which have rewriting actions, because main aim is to hinder the bypass threats towards an SDN firewall. The flow paths which consist of rewriting flow entries shifted flow paths. Those shifted flow paths compose a graph named Shifted Flow Path Graph. Also, the rules in a firewall build an Authorization Space. When detecting the conflicts between the firewall policy and flow policies, compare the Deny Authorization Space and the Sifted Flow Path Space.

C. Conflict Detection and Resolution:

Before detecting conflicts, the Deny Authorization Space and the Shifted Flow path Space are to be considered. For each rule in the Deny Authorization Space and the tracked space of each shifted flow path, detect whether they have intersections, if so, claim that there is a conflict between the firewall policy and the flow policies.

To resolve such conflicts, remove the entire flow path in the network or refuse to insert the flow entry that could cause conflicts. Then block the conflicting part of a flow path by inserting corresponding deny rules with a higher priority. For
example, consider a flow path having a source address 100x and a destination address 110x. The firewall deny rule is "101x→ 11xx : DENY". Therefore, this rule is conflicting with the flow path. To resolve this conflict, a new flow rule "1001 → 111x: DENY" is inserted in the ingress switch of the flow path and another new flow rule "101x → 1100: DENY" is added in the egress switch to block the conflict part of the flow path.

1) Adding New Firewall Rules:
Adding new rules to the firewall may cause conflicts between firewall policy and flow policies. If the new rules are with the actions other than deny, they will not cause bypass threats. So more focus should be on deny rules. Before detecting the conflicts, check the Deny Authorization Space first. Detect the new deny space introduced to the firewall by checking the overlapping relationships with other deny rules. Then, get the tracked space of the Shifted Flow Path Space and then check the conflicts between the new Deny Authorization Space and the tracked space.

If there are still conflicts, by adding new deny rules to the ingress switch and the egress switch of the flow path conflicts can be resolved. If the new inserted firewall rule introduces new deny space, there may be conflict flow paths in network due to rewriting the content of packet header fields. Therefore, the tracked space records the source and destination of the flow path. It is easy to detect the conflicts by comparing the tracked space with the firewall authorization space directly. If the tracked space is smaller than the firewall deny authoritarian space, this inserting request is rejected. But if the tracked space is bigger than newly introduced deny space in the firewall, only block the conflicting part of the flow path.

2) Adding New Flow Entries:
When network applications or controllers insert new flow entries to the flow tables, they may introduce new conflicts with the firewall policy. Before checking the conflicts, update the Shifted Flow Path Graph, because new inserted flow entry may change current flow paths and/or create new flow paths, which may introduce new conflicts. Once the conflicts are detected, different from adding new firewall rules, proposed conflict resolution solution only need to block the conflicting part of the flow path at its ingress switch. If the tracked space is smaller, the request of adding new flow entry will be refused directly.

VI. CONCLUSION
In this work, we addressed the challenge of building a reliable SDN firewall. In our approach, 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. During the conflict detection, the rule dependencies in both flow tables and firewall policies are considered. Furthermore, our approach provides a fine-granted conflict resolution. Finally, we implemented our security-enhanced SDN firewall application of FLOWGUARD in Floodlight [18]. Our experiment showed that our application can effectively and efficiently prevent bypass threats in OpenFlow networks.

REFERENCES
[16] Nate Foster, Micheal J. Freedman, Arjun Guha, Rob Harrison, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Mark Reitblatt, Jennifer Rexford, Cole Schlesinger, Alec Story and David Miller, Languages for Software Defined Networks.