Journal of Contemporary Issues in Business and Government Vol. 26, No. 03, 2020
https://cibgp.com/

P-ISSN: 2204-1990; E-ISSN: 1323-6903
DOI: 10.47750/cibg.2020.26.03.011

FLOWGAURD APPLICATION IN FLOODLIGHT FOR SECURED AND RELIABLEECURED SOFTWARE DEFINED NETWORKS

[1]G Harika Assistant Professor,
harikagora@gmail.com,
[2]Dr. J Rajaram Professor,
drrajaram81@gmail.com,
[3]S Raju Assistant Professor,
srajunayak@gmail.com,
[4]E Muralidhar Reddy Assistant Professor,
krishna81.reddy@gmail.com,
Department of CSE Engineering,
Pallavi Engineering College,
Kuntloor(V),Hayathnagar(M),Hyderabad,R.R.Dist.-501505.

Abstract: - Software-Defined Networking (SDN) provides network-wide access to programmers and direct control from a theoretically centralized controller over the underlying switches. SDN proposes a positive path for the Internet to grow in the future. However, SDN has several modern protection problems as well. How to develop a stable firewall programme for SDN is a vital task for them. Since the stateless property of the Open Flow-based SDN firewall lacks audit and monitoring mechanisms, current SDN firewall implementations may also be easily bypassed by rewriting the switch flow entries. Focusing on this hazard, by testing flow space and firewall authorization space, we implemented a novel approach for dispute detection and resolution in Open Flow focused firewalls. Unlike Fortnum, based on the whole flow paths inside an Open Flow network, our method will verify the contradictions between the firewall rules and flow policies. Finally, for flow tables and firewall guidelines, we introduced intra-table dependency testing.

Keywords: - Networks Identified Applications, Firewalls, and Space Analysis Header.

I. INTRODUCTION

It is an arduous process to run and manage a computer network. Network operators need to configure each individual network system separately from a heterogeneous set of switches, routers, middle boxes, etc., to communicate the appropriate high-level network policies, utilizing vendor-specific and low-level commands. Networks are dynamic in addition to configuration complexities, and operators have little or no tools to react automatically to network incidents. In such a constantly evolving climate, it is often difficult to implement the necessary policies. Network switches become basic forwarding machines with the isolation of the control plane from the data plane that lays the foundation for the Software Specified Networking model, and control logic is applied in a logically centralized controller.

An innovative network architecture implemented at Stanford University is Software Based Networking (SDN). This helps programmers, by machine engineering, to monitor and identify networks, which makes it known as advancement in the field of networking. As the central SDN technology, Open Flow (OF) [1] is a modern paradigm of network transfer that distinguishes network access and flow features. Users can monitor the activity of packets on networks in this model by integrating flow inputs into the switches. Switches and routers are implemented in a conventional network data plane and control plane, while SDN decouples those two flights. In an SDN, the control plane monitors the flow tables in the switches by utilizing a modern technique named the Open Flow protocol. The control plane, in this sense, understands the unified control over the whole network. For specialized work, a controller can compute the shortest flow paths and monitor the forwarding actions made by the switches. A device, a virtual machine, or a physical server might be the controller [2].

SDN has two critical functions. Next, an SDN distinguishes the control plane from the data plane (which determines how to manage the traffic) (which forwards the traffic according to decisions that the control plane makes). Second, the control plane is consolidated by an SDN, meaning different data plane components are managed by a single machine control programme. The SDN control plane maintains direct control over the state of the data plane elements (i.e., routers, switches, and other middle boxes) of the network using a well-defined application programming interface (API). A popular example of such an API is Open Flow. An Open

P-ISSN: 2204-1990; E-ISSN: 1323-6903
DOI: 10.47750/cibg.2020.26.03.011

Flow transfer has one or more packet-handling rules tables. Each law suits a traffic subset and conducts those traffic acts that conform to a rule; activities involve falling, routing, or flooding. An Open Flow switch will act like a router, switch, firewall, network address converter, or anything in between, depending on the rules installed by the controller application [3, 17].

While SDN offers many advantages in network growth, it also poses several new protection challenges. How to develop a stable firewall programme for SDN networks is one such problem. An important restriction of Open Flow is that it is stateless. For instance, if a host or network system sends a flow to the network, the controller will review only the first packet of the flow, while the resulting packets will be transmitted immediately by the switches without any exploration. The controller still has no audit or monitoring system set up for flows. Therefore, the current SDN firewall programme could easily be bypassed by intentionally adding the flow entries with rewriting operations [4].
A comprehensive methodology for dispute identification and resolution in SDN firewall is applied to resolve such a hazard by testing flow space and firewall permission space. To figure out how the flow paths interfere with the firewall rules, first browse the flow paths in the whole network and verify them against all firewall denial rules. Then, in the firewall guidelines or the flow tables, display various dispute mediation techniques according to different activities. Considering that the address space of a flow route can vary from the address space of the conflicting firewall laws, a mechanism was implemented to insert unique blocking flow input into the entry port or the exit port (they clearly identify the IN (Entry) and OUT traffic coming in) (Egress). Physical ports are not applied to them. They just refer to where the traffic starts from and to where it goes. It's all virtual, which is why the flow route is named entry and exit instead of 'incoming' and 'outgoing' ports). The firewall programme may use this tool to block packages that are in dispute with the firewall rules without interrupting any usual packages. A monitoring framework for flows can be developed by building and preserving a shifted flow graph, thereby addressing the bypass issue fundamentally.

2. EMPLOYMENT CONNECTED

The protection challenges in SDN have drawn further interest lately with the rapid advances in SDN techniques.

Son et al., [1] have implemented FLOVER, a model control framework that verifies that the flow policy aggregate instantiated inside the Open Flow network is not in violation of the network protection policy. Their framework identifies faults that lead to invalid and invisible paths, but firewall rules are not regarded.

R. Sherwood et al. [2] [3] suggest FlowVisor, which allows stable network functions to be segmented into separate virtual machines by segmenting or splitting network access. A self-consistent OF application is regulated by each network domain, which is architected to not interact with OF applications that rule other slices of the network. In this way, the protection of Open Flow is cast as a property without intervention. However, the issue persists that a network provider can also choose to instantiate network protection restrictions that must be implemented within the slice, even inside a specified network slice.
Alou et al., [4] suggested verification toll for the firewall that takes input of a firewall policy and a specified property, then outputs if the property is fulfilled by the policy. Using decision diagrams, Liu developed and applied a verification algorithm and checked it on both real-life firewall policies and large-scale synthetic firewall policies.

Alou et al., [5] did some work on modeling protection strategies for firewalls. These studies do not, however, discuss the complex existence of flow rules in networks specified by software.

E. Al-Shaer et al.,[6] conducted some Flow Checker-related work that encodes OF flow tables into the Binary Decision Diagram (BDD) and uses model testing to validate protection properties and verify that the OF network is divided into equivalence groups to effectively check for invariant property violations [7]. However, intermediate behavior such as set and got commands are not discussed directly by these programmers.

P-ISSN: 2204-1990; E-ISSN: 1323-6903
DOI: 10.47750/cibg.2020.26.03.011

M. Canine et al., [8] suggested the usage of symbolic execution by Good to check conformity with OF applications. Such approaches to route discovery do not, however, scale well for broad applications.

N. McKeon et al. [9] implemented the OpenFlow transfer principle and used it in numerous implementations, such as NOX (Network operating system) and Flow Visor (A network virtualized layer). The work done on Open Flow switches did not discuss the issues of conflict analysis and testing of the concept; instead, it demonstrated the simple Open Flow model design and how it can be used to provide the physical network with conceptual separate networks.

R. Sherwood et al.,[10] implemented the virtualization layer of the network, where a development network is broken into several virtual networks that concurrently execute multiple experiments, each with its own forwarding decision.

M. Uh, Casado et al,. [11][12] Recommended a modern framework for corporate network defense. The SANE [11] defense layer suggests a fork-lift (clean-slate) solution to corporate network security enhancements that implements a centralized server, i.e., a domain controller, to authenticate all network elements and to grant access to services in the form of capabilities implemented at each switch. Ethane [12] is a more realistic and backward-compatible SANE instantiation that does not need end hosts to be changed. Alongside standard network switches, Ethane switches live and connect with the unified control that implements policy. Both experiments may be used as catalysts for Open Flow and software-defined networking to evolve.

Went et al.,[13] suggested as the first line of protection, PermOF, a fine-grained authorization method, to extend minimum privilege to applications. They summarized a list of 18 permissions to be implemented in the controller's API entry.

3. GAPS OBSERVED

SDN was only launched a number of years back, as a modern network paradigm. Because it enables network programmers to work directly with switches on the networks, it poses a number of protection challenges. The following are the study holes identified:

- The firewall rules do not consider the discovery of errors leading to invalid and unseen routes in current approaches [1].
- Existing Open Flow structures in software-defined networks are stateless and do not resolve the complex existence of flow rules [4] [5].
- The dispute between the separate implementations in the current SDN controllers [15] is not regarded.
- Firewalls or other protection applications may be quickly bypassed in the current architecture [14] by introducing intentional flow tables.
- New approaches [14] do not take into consideration the intra-table dependence between firewall rules and flow entries.

In the current framework, there is no audit or monitoring methods in the controller package for flows.

For large applications, current systems [8] do not scale well.

The above problems have been established as priorities for the planned work in this respect.

4. PROPOSED WORK'S

Software Defined Networking (SDN), offers programmers with network-wide insight and power from a logically unified controller over the fundamental switches, not only has an immense effect on computer network growth, it also provides a promising way for potential internet creation. There are several threats to protection that fall into the frame. How to develop a stable firewall programme for SDN is one such problem. The new SDN firewalls can be quickly bypassed by rewriting the flow entries in switches due to the stateless property of the Open Flow-based SDN firewall and the absence of monitoring and audit mechanisms. In Open Flow-based firewalls, a comprehensive approach to dispute identification and resolution by testing flow space and firewall permission space is implemented to concentrate on this hazard. Unlike Fortnum [14], based on the whole flow paths inside an Open Flow network, the proposed solution will verify the contradictions between the firewall rules and flow policies. It is also important to incorporate intra-table dependence testing for flow tables and firewall laws. Finally, the implementation of the proposed solution is discussed as a proof-of-concept

P-ISSN: 2204-1990; E-ISSN: 1323-6903
DOI: 10.47750/cibg.2020.26.03.011

and preliminary findings would prove that the approach will successfully deter bypass attacks in actual Open Flow networks.

5. METHODOLOGY

A. Header Space Analysis:

The suggested algorithm for dispute identification and settlement is focused on Header Space Research. HSA offers a standardized and protocol-agnostic architecture of the network that utilizes a geometric packet processing model. A header is specified as a {0, 1} L space point called the header space (L is the packet's length (in bits)). Network boxes are modeled using Switch Transfer Function T, which moves the obtained header h to a packet header set on one or more output ports in the input port:

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

B. Shifted Flow Space and Authorization Space:

In order to verify if the firewall rules clash with Open Flow switch flow tables, all packets should be monitored and all destinations reached by the packets should be determined and the header space at each destination should be calculated. It is important to equate the source address and destination address in the header space of a flow path with the address space obtained from the firewall policy. The flow laws are known to interfere with the firewall protocol if they have an intersection.

The firewall rule usually consists of 5 fields: source address, source port, address of destination, port of destination, and protocol. A flow path's input header space is made up of three fields: source address, source port, and protocol. There are two fields in the output header space of the flow path: the destination addresses and the destination terminal. The source and endpoint of a traffic flow path can be defined by the entry and exit space that constructs a monitored space of a flow path. A flow graph is generated by all flow routes, and is called the net plumbing graph [15].

The recommended approach to dispute resolution only takes into consideration the flow pathways that have rewriting actions, as the key goal is to obstruct the challenges to an SDN firewall bypass. The flow paths that comprise of flow entries rewriting moved flow paths. A network called Shifted Flow Path Graph composes such shifted flow paths. Additionally, the rules in a firewall provide an Authorization Area. Compare the Refuse Permission Space and the Sifted Flow Route Space while finding contradictions between the firewall protocol and flow policies.

C. Identification and Settlement of Conflict:

The Reject Permission Space and the Moved Flow path Space need to be considered before detecting conflicts. Detect if they have intersections with each law in the Refuse Permission Space and the monitored space in each moved flow direction, if so; say that there is a discrepancy between the firewall rules and the flow policies.

Delete the whole flow route in the network to address those disputes or fail to introduce a flow entry that would create conflicts. Then, by adding corresponding deny rules of higher priority; block the conflicting portion of a flow path. Consider, for instance, a flow path having a 100x source address and a 110x destination address. 101 xs to 11xx: DENY" is the firewall deny rule." Hence, the flow direction clashes with this law. A new flow rule '1001 ⁇ 111x: DENY' is inserted in the flow path entry switch to settle this dispute, and a new flow rule '101x ⁇ 1100: DENY' is inserted in the egress switch to block the flow path conflict portion.

1) Introducing Additional Guidelines for Firewalls:

Adding new rules to the firewall can trigger firewall policy and flow policy conflicts. If the current laws do not deny conduct, they may not establish threats to circumvent them. So, there should be more focus on refusing laws. Second, search the Refuse Permission Space before finding the conflicts. By testing the conflicting relationships with other deny laws, detect the latest deny room added to the firewall. Then, have the Moved Flow Direction Space monitored space and then search the disputes between the current Reject Permission Space and the tracked space.

If disputes still remain, it is possible to overcome the flow path conflicts by applying new reject laws to the ingress switch and the egress switch. If new deny space is added by the new inserted firewall law, dispute flow paths can occur in the network due to the rewriting of the packet header field information. The monitored space tracks, thus, the source and destination of the flow direction. The disputes can be easily identified by closely contrasting the monitored space with the firewall permission space. If the monitored space is less than the refused oppressive space by the firewall, this proposal for inclusion is declined. But if the

P-ISSN: 2204-1990; E-ISSN: 1323-6903
DOI: 10.47750/cibg.2020.26.03.011

monitored room in the firewall is greater than the recently added deny space, block the conflicting portion of the flow path only.

2) Introducing New Entries in Flow:

As network applications or controllers insert new flow entries into the flow tables, new firewall policy disputes can occur. Adjust the Shifted Flow Path Graph before testing the conflicts, so the latest added flow entry will alter existing flow paths and/or build new flow paths, which can contribute to new conflicts. Once the disputes are found, the desired dispute resolution approach only has to obstruct the overlapping section of the flow path at its entry switch, as opposed to introducing additional firewall laws. If the monitored room is lower, the appeal for additional flow entry to be inserted would be explicitly denied.

6. CONCLUSION

We tackled the task of designing a secure SDN firewall in this work. In our method, binary vectors are first represented by the source and destination addresses of firewall rules and flow entries. Then, by contrasting the moved flow space and refusing firewall authorization space, contradictions between firewall rules and flow rules are reviewed. The rule dependencies in both flow tables and firewall rules are known throughout the identification of disputes. In addition, a fine-granted dispute settlement is given by our method. Finally, in Floodlight [18], we deployed our FLOWGUARD security-enhanced SDN firewall programme. Our experiment shows that our programme would avoid bypass attacks on Open Flow networks easily and efficiently.

References

[1] Son, S., Shin, S., Yegneswaran, V., Porras, P.: Model Checking Invariant Security Properties in OpenFlow. In: Proc. of ICC 2013, pp. 2–6, 2013.

[2] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network Be the Testbed. In Proceedings of the Use nix Symposium on Operating System Design and Implementation (OSDI), 2010.

[3] OpenFlowHub. BEACON. http://www.openflowhub.org/display/Beacon.

[4] A. Liu. Formal Verification of Firewall Policies. In Proceedings of the International Conference on Communications (ICC), 2008.

[5] A. Liu and M. Gouda. Diverse firewall design. IEEE Transactions on Parallel and Distributed Systems, 2008.

[6] E. Al-Shaer and S. Al-Haj. Flow checker: configuration analysis and verification of federated open flow infrastructures. In Proceedings of the 3rd ACM workshop on Assumable and Usable Security Configuration, 2010.

[7] A. Churched, W. Zhou, M. Caesar, and P. B. Godfrey. Overflow: Verifying Network-Wide Invariants in Real Time. In Proceedings of ACM Sitcom Hosted Workshop, 2012.

[8] M. Canine, D. Venango, P. Pereˆs´ıni, D. Kosti´c, and J. Rexford. A NICE Way to Test OpenFlow Applications. In Proceedings of NSDI, 2012.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev., 38(2):69–74, 2008.

[10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flow visor: A network virtualization layer. Technical Report OpenFlow Technical Report 2009-1, Deutsche Telekom Inc. R&D Lab, Stanford University, Nicer Networks, October 2009.

[11] M. Cased, T. Garfunkel, M. Freedman, A. Avella, D. Bone, N. McEwen, and S. Shankar. SANE: Protection Architecture for Enterprise Networks. In Proceedings of the Use nix Security Symposium, 2006.

[12] M. Cased, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shankar. Ethane: Taking Control of the Enterprise. In Proceedings of ACM SIGCOMM, 2007.

[13] Wen, X., Chen, Y., Hu, C., Shi, and C.: Towards a Secure Controller Platform for OpenFlow. In: Proc. of HotSDN 2013 (2013).

[14] Porras, P., Shin, S., Yegneswaran, V., Fong, M.: A Security Enforcement Kernel for OpenFlow Networks. In: Proc. of HotSDN 2012, pp. 123–125 (2012)

[15] Kazemian, P., Chang, M., Zeng, H.: Real Time Network Policy Checking using Header Space Analysis. In: Proceedings of the Symposium on Network Systems Design and Implementation (NSDI), pp. 4–6 (2013).

[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.

[17] Nick Feamster, Jennifer Rexford, Ellen Zeruga, The Road to SDN: An Intellectual History of Programmble Networks.

[18] Floodlight: Open SDN Controller. http://www.projectfloodlight.org.