Grenze International Journal of Engineering and Technology, Jul 2019

Advanced Flow Monitoring Technique for Intrusion Tolerance in SDN

Manu B[1] and Sneha M[2]
[1-2]Department of Computer Science, R.V College of Engineering, Bangalore, India
Email: manubk2003@gmail.com, sneham@rvce.edu.in

***Abstract*—It is important for a system to build security features to prevent attacks and protect the infrastructure. Since it is very hard to achieve full protection, the notion of intrusion tolerance came into existence. In intrusion tolerant architectures, every single intrusion is not prevented. The intrusions are allowed but tolerated. The system has mechanisms which will prevent the intrusion to turn to a system failure. There are a lot of papers in classical IP-based networks regarding intrusion tolerant systems, but a very few introduce the concept of intrusion tolerance in software define networks (SDN). Software-defined networks (SDN) is an evolving architecture in which control plane is detached from the data plane. The controller, which constitutes the SDN control plane, can control the entire network. This centralized view of the network with the ability to program the network through applications can be used for developing intrusion tolerant architecture.**

The implementation discussed here is based on the flow monitoring method [1] for intrusion tolerance in SDN. Intrusion Detection Module (IDM) is developed on top of MD-SAL [2] in the Opendaylight SDN controller. IDM uses Openflowplugin module to request flow statistics from the switch and monitors packet counts at different intervals. It programs higher priority flow with the same match criteria and action as sent to controller if the rate exceeds the threshold value. This higher priority flow is programmed with hard timeout of 5 seconds which prevents overburdening of sending a lot of packets to controller. IDM also performs deep packet inspection when the packet reached controller. The packets for the stream are successfully dropped if the intrusion is from a known sender. This is done by matching on the source and destination ports. For the packets from an unknown source, a new flow is added with IP address as match and an action of drop. This prevents further intrusion for all the packets sent from the unknown sender and thereby protecting the system from failure

***Index Terms*— SDN, Intrusion tolerant Architecture, Opendaylight.**

I. INTRODUCTION

In traditional network architecture, it requires a lot of manual work to make even small changes in the network. The configuration changes must be done on thousands of network devices scattered across the network. These network devices could be vendor specific and there could be interaction problems with other third-party vendor devices. Agility is equally important in a rapidly changing network along with performance and stability. Since SDN [3] has moved intelligence to central control, it has become easier to make network changes in an environment where network demands change quickly. Along with solving the problems of traditional network

Grenze ID: 01.GIJET.5.2.503_1

architecture, SDN also provides a programmable interface so that any vendor can develop any services. This flexibility and agility of SDN technology helps even network managers to make changes in the network easily using SDN programs. The different advantages of SDN architecture is used in this project to develop intrusion tolerant architecture using flow monitoring technique.

As the number and variety of cyber threats increase, it becomes more critical to share intelligence information in a fast and efficient manner. However, current cyber threat intelligence data do not contain enough information about how to specify countermeasures or how institutions should apply countermeasures automatically on their networks. The countermeasures must be taken carefully without impacting the remaining data traffic. The down time for traffic other than the malicious traffic should be kept as minimal as possible. The SDN intrusion tolerant architecture must be developed as complete solution without depending on other agents. This provides ease of use and flexibility to the solution. Intrusion must be detected using the rate of traffic from the flows. Rate-limiting must be done on the switch once the controller detects the intrusion. Depending on the source, either all the packets are dropped, or a specific traffic flow is not allowed. A trusted source could have an application misbehaving which doesn't require all the packets to be dropped. The specific application must be blocked. For an untrusted source, all the packets could be dropped. Rate-limiting also must be done for the packets to the controller from switch. The controller should not be overburdened with the forwarding plane traffic. A few packets are required to detect intrusion. The remaining packets must be blocked from the switch.

II. Related Work

There are a lot of papers in classical IP-based networks regarding intrusion tolerant system, but a very few introduce the concept of intrusion tolerance in software define networks (SDN) [4]. The strategies include maintaining a suspicious source list [5] where the attacker IP is added to the list. Flow table is modified based on the suspicious source list. One among these studies consider that attacker will scan to understand the behavior of network before attack. The scan tool can detect services hosted on remote host, the operating system it uses and even the version of remote host. After the information is obtained, the attacker locks a specific object and then launch a series of attacks behavior.

In another strategy, a Moving Target Defence (MTD) [6] is established which maintains attribute diversity and constantly changing network system which makes it difficult for hackers to detect targets. The attributes which can vary in random includes host identity, port number, IP address, route path and so on. This introduces a mechanism called OpenFlow Random Host Mutation (OFRHM) where controller assigns a random virtual IP which is translated to/from the real IP of the host. It develops an optimized moving target defence architecture where routable short-lived virtual IP addresses (vIP) which changes randomly is used.

Another method uses the difference between the average recorded behavior [7] and the current observed behavior. In this strategy, a profile is generated first. It takes time to generate this profile. The data log is the main component in this method and is used for comparing the unexpected behavior. It is generated by monitoring host or a network device. SDN can make measures quickly when the network is being attacked. There are other studies where machine learning algorithms [8] is used to detect malicious activities. Self-organized maps (SOM) [9] is used in one of these studies for detecting unauthorized activities.

Most of the available studies uses a separate sampling agent and route the traffic to the agent using SDN. None of them provides a complete SDN solution. In a service provider network or a cloud platform, SDN controller has complete information regarding the trusted sources. For a service provider network, the subscriber information is available to the controller. Similarly, for a cloud platform, the IP address of the virtual machine is known to the controller. This is not being utilized currently.

III. System Design

The proposed system uses both the advantages of Intrusion tolerance and SDN. Opendaylight [10] is used as the SDN controller and Open Switch [11] is used as the switch. IDM is developed, and it monitors the packet count for flows. The flows are monitored using OpenFlowplugin [12] module in Opendaylight. The OpenFlowplugin supports OpenFlow 1.0 and 1.3.x and it is evolving based on new OpenFlow protocol [13] studies. The plugin is based on the Model Driven Service Abstraction Layer (MD-SAL) architecture.

IDM gets the flow statistics response through MD-SAL which wires IDM and OpenFlowplugin. MD-SAL provides common infrastructure services to applications and plugins. It currently provides data store services, RPC support, notification subscription and service publication. The packet rate is calculated by requesting flow statistics from the switch and monitoring packet counts at different intervals. The difference between packet

counts at different intervals is calculated. The difference is monitored for multiple number of times. If threshold exceeds multiple number of times, then the IDM initiates the action. IDM installs a new flow with a higher priority and hard timeout of 5 seconds. It has an action of sent to controller with the same match as the flow where intrusion is detected. This results in next packets to get sent to controller.

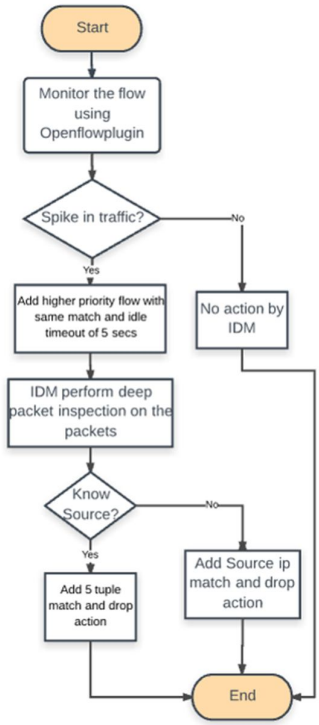

Figure 1. Flowchart for intrusion tolerant technique implementation

Controller will perform deep packet inspection on the received packets as show in figure 1. The source IP or MAC is used to detect whether the packet is from unknown source. Known source is the IP or MAC which is familiar to the environment (Cloud or Telecom). It could be a known customer IP or VM IP. If the packet is from a known sender, then packet is dropped for that specific traffic flow. This is done by adding a new flow with 5-tuple match [14] and a drop action. If the packet is from an unknown source, a new flow is added with match as subscriber IP and an action as drop. This will drop all the packets sent from the unknown sender and thereby protecting the system from failure.

IV. EVALUATION

This section describes in detail the data set used, evaluation metrics and the evaluation of the model [1]. Firstly, an insight to the data set used in the model is provided, then information about the evaluation metrics is described and finally the results and inference of the results are detailed.

A. Experimental Data Set

The different experimental datasets include fine tuning of the below parameters.

- Time interval to query the flow statistics from the switch
- The hard timeout configured in the flow which sent packets to controller
- The increment in the packet count during the query interval to detect intrusion

The interval to query the flow statistics is relevant since this determines the speed at which intrusion can be detected. If the query interval is too high, then the time to detect the intrusion increases. Thus, it delays the programming of drop flow. If the query interval is too low, then it can have performance impact.

The hard timeout configured in the flow determines how many packets are sent to controller. If the time configured is more, then more number of packets are sent to controller which can impact performance. And if the time configured is very less, there is a chance of controller missing the packets.

The increment in packet count must be determined based on the bandwidth allocated to the known source. For an unknown source, the packet count increment of even a single packet would be enough.

B. Evaluation Metrics

Evaluation metrics include mainly how the SDN controller responds to the intrusion detection. It must install the drop flow immediately as it detects the intrusion. The evaluation metrics therefore includes the time to respond to the intrusion. This is the difference between the time at which controller installs drop flow and the time at which the controller detects the spike in traffic.

C. Test Environment

OpenStack [15] integrated with ODL is used as the test environment. The traffic is sent from the Virtual Machine to the OVS. Two Virtual Machines are brought up using Openstack as show in figure 2. One acts as source of traffic and the other is used as destination. Openstack itself is deployed on a VM using VirtualBox [16]. Controller runs in the laptop which communicates with OVS and Openstack inside the VM.

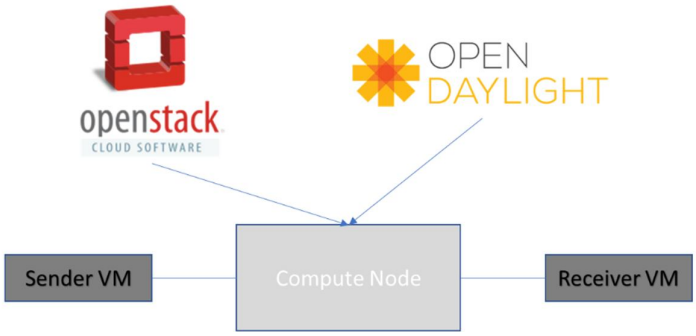

Figure 2. Test Environment

Traffic is sent from the source VM to test known source. To test unknown source, a virtual interface is created inside the VM with a different IP address. The traffic is then sent from this virtual interface to test the unknown source.

D. Inference from the results

The test is done with two Virtual Machines and the traffic is sent from the source VM. The hard time out of 5 seconds is configured. 20 seconds query interval is configured to avoid frequent polling of the flow statistics from the switch. Controller detected intrusion in 20 seconds as the query interval is configured to 20 seconds. It then immediately configures the flow with hard time out and the action to send to controller. The subsequent packets are thus sent to controller. The drop flow is configured immediately by the controller after getting details from the packet received. The hard time out and query interval must be fine-tuned again by trying out different possible values to optimize the detection of intrusion. The time to be configured would be a trade-off between the controller performance and the detection of intrusion.

The query interval is configured with different values and then tested. Figure 3 shows the graph plotted for different values of query interval and time interval to program drop flow. The intrusion detection time is reduced by reducing the query interval. The test is done in a single node controller setup with traffic sent at a higher rate from source VM to destination VM.

The number of packets allowed before intrusion was calculated using the packet counts associated with each flow. Packet count was calculated before and after sending the stream of packets from the test sender VM. Figure 3 shows the graph plotted for different values of time interval to program drop flow and number of packets allowed before stopping intrusion. It was found that the number of packets allowed before stopping intrusion increased with increase in time to program drop flow.

Figure 4 shows the graph plotted for number of packets allowed before stopping intrusion at different attempts. This graph is also plotted against different statistics query interval timings. It was found that the number of packets allowed were almost same at different attempts for statistics query interval time of 1 sec. But it varied for different attempts in case of statistics query interval time of 10 secs. This difference is due to the time taken for detecting the intrusion and programming the drop flow.

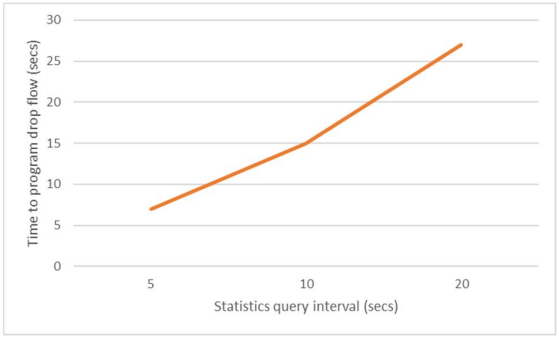

Figure 3. Graph plotted for different values of query interval and drop flow programming interval

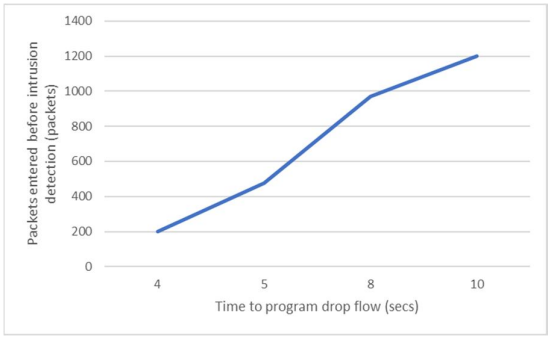

Figure 4. Graph plotted for different values of time interval to program drop flow and number of packets allowed before intrusion

The statistics query was done at every 10 secs for the second case. So, the number of packets allowed before stopping intrusion depends on the time at which drop flow is programmed. This in turn depends on the time remaining for the next statistics query. Since there is a larger gap of 10 secs in case of second case, the drop flow is programmed at next statistics query and after detecting intrusion. This explains the random graph form for statistics query interval of 10 secs and almost a straight line in case of statistics interval of 1 sec.

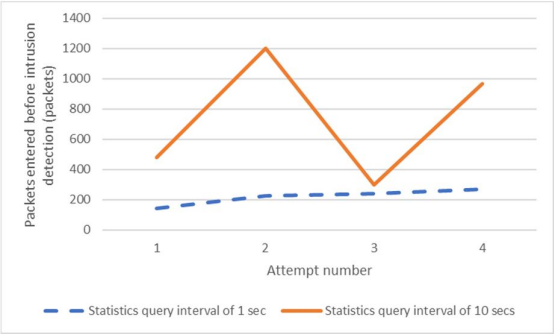

Figure 5. Graph plotted for different values of number of packets allowed before detecting intrusion and time taken to program drop flow

V. Conclusion And Future Work

ODL controller detects intrusion depending on the query interval configured in the controller. It then immediately configures the flow with hard time out and action as sent to controller. The subsequent packets are thus sent to controller and the drop flow gets configured immediately by the controller after getting details from the packets received. This dropped further packets from the source of intrusion.

The increment in packet count used for intrusion tolerant SDN architecture is a fixed value. It must be a dynamic value which varies according to the bandwidth allotted to each subscriber. As part of future work, the model can be enhanced to store the values against each subscriber.

The model implementation is performed on a single node ODL controller. The model can be run in a three-node cluster with a scaled environment. With the increase in the number of compute nodes used, the three-node cluster can perform better. The compute nodes connectivity can be distributed across the different controller nodes. The intrusion tolerant SDN architecture can be enhanced to scale up to the number of subscribers.

References

[1] B. Manu and A. K. Koundinya, "Intrusion Tolerant Architecture for SDN Networks Through Flow Monitoring," 2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bangalore, 2017, pp. 1-5.

[2] M. Paliwal, D. Shrimankar and O. Tembhurne, "Controllers in SDN: A Review Report," in IEEE Access, vol. 6, pp. 36256-36270, 2018.

[3] K. Raghunath and P. Krishnan, "Towards A Secure SDN Architecture," 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bangalore, 2018, pp. 1-7.

[4] Juma Ibrahim, Slavko Gajin, “SDN-Based Intrusion Detection System”, 2017 INFOTEH-JAHORINA Symposium, Jahorina, 2017, pp 2-7.
[5] P. J. Chen, Y. W. Chen, "Implementation of SDN based network intrusion detection and prevention system," 2015 International Carnahan Conference on Security Technology (ICCST), Taipei, 2015, pp. 141-146.
[6] P. Kampanakis, H. Perros and T. Beyene, "SDN-based solutions for Moving Target Defense network protection," Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, 2014, pp.1-6.
[7] M. Liyanage et al., "Enhancing Security of Software Defined Mobile Networks," in IEEE Access, vol. 5, pp. 9422-9438, 2017.
[8] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," 2017 Seventh International Conference on Emerging Security Technologies (EST), Canterbury, 2017, pp. 138-143
[9] Damian Jankowski, Marek Amanowicz, “Intrusion Detection in Software Defined Networks with Self-organized Maps”, 2015 Journal of Telecommunications and Information Technology, Poland, 2015, pp 7-9.
[10] Opendaylight - The Linux Foundation Projects (Accessed on 27/12/2018). [Online]. Available: https://www.opendaylight.org/
[11] Open vswitch. (Accessed on 27/12/2018). [Online]. Available: http://openvswitch.org//
[12] OpenDaylight OpenFlow Plugin:Main. (Accessed on 27/12/2018). [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Main
[13] OpenFlow. (Accessed on 27/12/2018). [Online]. Available: https://en.wikipedia.org/wiki/OpenFlow
[14] TCP Analysis and the Five-Tuple | Packet Foo | Analyzing network packets. (Accessed on 27/12/2018). [Online]. Available: https://blog.packet-foo.com/2015/03/tcp-analysis-and-the-five-tuple/
[15] O. Tkachova, M. J. Salim and A. R. Yahya, "An analysis of SDN-OpenStack integration," 2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T), Kharkiv, 2015, pp. 60-62.
[16] VirtualBox. (Accessed on 27/12/2018). [Online]. Available: https://www.virtualbox.org/