DETECTING MALICIOUS CONTROLLER BASED ON THREAT VECTORS IN SOFTWARE DEFINED NETWORKS

R. Rajalakshmi[1], L. Leo Prasanth[2], Dr. E. Uma[3]

[1]M. Tech, [2]Research scholar, Department of IST, CEG(ANNA UNVERSITY), Chennai, TamilNadu, India
[3]Assistant Professor(SL. Gr), Department of IST, CEG(ANNA UNVERSITY), Chennai, TamilNadu, India.

Abstract - *Software Defined Networks is a developing field in computer networking. It is a new architecture that splits the data plane and control plane. Control plane is a vital part of the SDN design, so it's very significant to offer proper attention to style any SDN controller. As an outcome of centralized action of the control plane, it's far essential to find the presence of malicious control plane in SDN. Malicious control plane alludes to the condition where at least one of the controllers in SDN are compromised through malwares, bringing about deviation from the ordinary control plane behavior. Our task is finding an approach to identify a malicious controller. Dynamic creation of varied topologies is required to research data modified in packets. It includes i) Finding four threat vectors that represent malicious controller. ii) Creating dataset from data plane packet logs. iii) Six Open Flow features that capture the threat vectors. iv) Machine - learning based recognition system for malicious control plane using random forest and decision tree classifier. Random forest shows the higher accurateness when compared to decision tree.*

***Key Words*: malicious controller, machine learning, open flow, software defined networking, Security.**

1. INTRODUCTION

Software Defined Networks is a developing field in computer networking. It permits network managers to modify a specified network rendering to changing client or business needs. It acts as a physical separation between network control plane and forwarding plane, wherever a control plane manages multiple devices.

SDN has some issues such as single point failure, the communication overhead between switches and controllers, more significant for security and also trust ability of the network control plane. It is an essential to find the presence of control plane which is compromised in SDN because of centralized control plane. For security concerns, SDN exchanges rules in firewalls with flow rules at distinct switches and it may enforce node level security.

The control plane becomes compromised due to the controller is promised by certain threats, subsequent in variation from standard control plane's behavior. The new results are developed for perceiving malicious controllers due to the absence of legitimate controllers.

The OpenFlow protocol can revolution in network activities have a subsequent variation in the preceding switch controller message. Using the OpenFlow specifications, switches are configured to work with the comparative outcomes to a legacy switch, without having to manually reconfigure the switch if the network varies. The open flow traces are wanted to find the existence of malicious controllers.

2. EXISTING METHODOLOGY

The existing system has established some security architecture in SDN. The lack of existing principles in emerging controller permits even a compromised switch to interrupt the complete control network. So, there is a difficult to detect the malicious switches. Attacks are possible due to the deployment of malicious controller applications. Due to the result, centralized way in SDN controllers becomes the actual goals for the intruders to gain contact the entire system. At least one controller in SDN are malicious by malwares, bringing out deviation from the ordinary control plane performance.

3. PROPOSED METHODOLOGY

3.1 Overview

Detecting malicious controller system provides more security than detecting compromised data plane devices. The proposed system is to represent the four different threat vectors are used to detect the malicious controllers. The vectors are 1) The malicious controller application will make the network intentionally slow down by delay the control plane reply. 2) It can establish flow rules in switches to diverge packets from the shortest route. 3) It identifies critical switches and establishes DROP actions which are from flow rules to refuse services from that switch. 4) It replicates the packets and sends them to unexpected destinations. Open flow packet traces are taken out from data plane for a definite time and open flow specific information are extracted. Malicious controllers are detected using open flow traffic. The threat vectors are detected using open flow traffic traces. Six features are drawn from the open flow traces. The packet-in packet-out proportion, packet-in packet-out divergence, switches association index, precedence frequency spike index, variance of drop action, timeout frequency spike index. These statistics is moved to a system where learning and classification are passed out from the Open Flow traces, *FlowMod*, *packet-in* and *packet-out* types of packets are filtered to compute six features. The network samples are classified using random forest and decision tree classifier as shown in the Figure. 1. Random forest classifier indicates the high accuracy when compared to decision tree classifier.

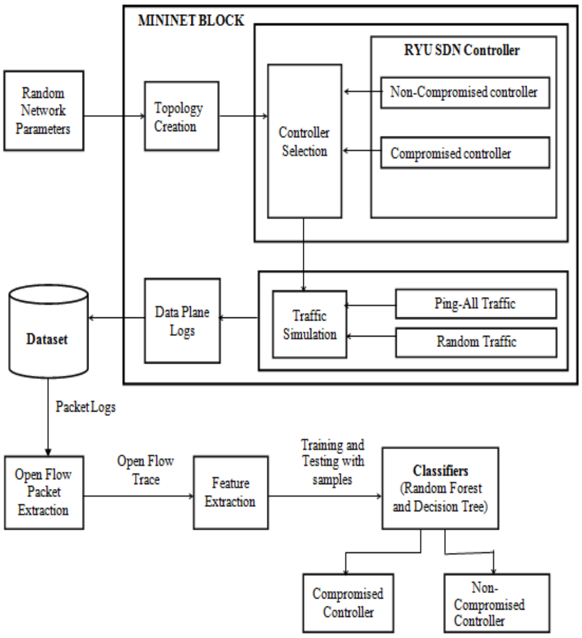

Figure. 1: Block diagram for detecting malicious controller

3.2 Features Extraction

1. Packet-in packet-out proportion:

Controller calculates the forwarding result and directs a FlowMod message come back to the switch to avert succeeding packet-in messages. The first packet goes to the flow is held using a packet-out message with essential actions.

A malicious controller make an effort DDoS attack may purposely delay the administrative process and also decrease the quality of service. PPP is characteristics that fraction of the amount of packet-in received messages per packet-out received message is calculated using Alg. 1.

Algorithm. 1: Packet-in Packet-out Proportion

Input: open flow Traffic
Output: PPP

1. Initialize globally, pin=0, pout=0, t=5ms
2. pcktprocessor method starts
3. t = 0
4. Till t arrive at its threshold
5. for every packet forwarded
6. pin++
7. for every packet gets handled and forwarded
8. pout++
9. PPP=pin/pout
10. Return PPP
11. pcktprocessor method ends

---

2. Packet-in Packet-out Divergence:

The controller replies to switch with the suitable action through packet-out messages. Malicious controller sends a suitable action along with the initial packet-out message and then sends another one packet-out with abnormal actions. PPD's calculated by the ratio of extra packet-out message to the total amount of packet-out messages are computed using Alg. 2.

Algorithm. 2: Packet-in Packet-out Divergence

Input: open flow traffic
Output: PPD

1. Initialize globally, pin=0, pout=0, t=5ms
2. pcktprocessor method starts
3. t = 0
4. Till t arrive at its threshold
5. for every packet forwarded
6. pin++
7. for every packet gets handled and forwarded
8. pout++
9. PPD=pin-pout/pout
10. Return PPD
11. pcktprocessor method ends

---

3. Switch Association Index:

The switch forwards packets according to the flow rules referred by the controller. When a packet comes, the switch checks for the suitable flow rules in its flow table. If the packet does not equivalent with any of the remaining rules, the switch directs a packet-in message to the controller. The number of flow rules installed is directly proportional to the number of switches considered in the network. A switch associated with the malicious controller usually has a bottleneck. Therefore, SAI is used as the adjustment of typical amount of flow rules for every port in a switch. Steps to compute SAI are given in Alg. 3.

Algorithm. 3: Switch Association Index

1: pcktprocessor method starts
2: Get S be the number of switches from topology
3: For every i in S
4: M_i = number of flows sent/degree of switch
5: M= M+M_i
6: for loop end
7: M = M/S
8: for every i in S
9: SAI_i = ((number of flows sent/degree of switch S_i) – M(Si))2
10: SAI = SAI +SAI_i
11: for loop end
12: SAI = SAI/S
13: Return SAI and pcktprocessor method ends

4. Variance of drop actions:

The non-responsiveness is understood using flow rules with drop actions. Packets are dropped by protocols and firewalls. It recognizes uncommon differences in amount of DROP actions distributed to several numbers of switches. In open flow, an unfilled action field in the message that is FlowMod indicates a DROP action in flow rule. VDA is computed using Eq.1.

$$VDA= \frac{\sum_{i=1}^{Ts}(mi-\mu)^2}{Ts} \qquad (1)$$

Where, $\mu=\frac{\sum_{i=1}^{Ts}(mi)}{T_s}$

T_s be the total amount of switches from messages that is FlowMod in packet traces which is open flow and m_i represents the number of messages that is FlowMod delivered to switch which id is i with DROP action

5. Priority Frequency Spike Index:

PFSI capture both spikes and frequency dips detected from the importance standards of the particular flow rules. Compromised flow rules incline to operate at higher importance in order to avoid packets similar with malicious rules. The occurrences of FlowMod messages remain comparatively low related to unaffected flows.

$$PFSI = \max((\max(np_{pi})-\mu), (\mu-\min(np_{pi})))$$

Where,

$$\mu = \frac{\sum_{i=1}^{Np}(np)}{Np_p} \qquad (2)$$

Np_p indicates the number of unique priority values from FlowMod messages with priority p_i is calculated using the Eq.2.

6. Timeout Frequency Spike Index:

Malicious flow rules purposes are incline to have a higher timeout values, such FlowMod messages arise at decrease frequencies. Malicious controller also set with the flow rules with lesser timeout values to often examine the packets via packet-in message. TFSI is computed using Eq.3.

$$TFSI = \max((\max(nt_{ti})-\mu), (\mu-\min(nt_{ti}))) \qquad (3)$$

In which,

$$\mu = \frac{\sum_{i=1}^{Ntt}(ntti)}{Nt_t}$$

Nt_t is the total amount of distinctive timeout values determined from FlowMod messages and nt_{ti} is the total amount of FlowMod messages with timeout t_i.

4. IMPLEMENTATION

Simulation set-up:

Mininet test system has been utilized to make SDN environment comprising of end hosts, switches, SDN controllers and controller applications. Mininet is made to run on ubuntu 18.04 working framework with various topologies. Switches are moved up to

OpenvSwitch and are arranged to utilize open flow v1.3.

Traffic is created using ping command indicating source, destination, number of packets and type of traffic. And also, using Iperf utility generate the traffic. In this system the RYU controllers are used for selecting controller design. Mininet is controlled the RYU controller.

4.1. Topology Creation

During this topology creation phase, creating the realistic virtual networks for organizing switches, controllers and hosts using Mininet. It is an open source software tool that is used to simulate an SDN and its well-suited controllers, switches and hosts.

It also supports for variation of topologies and confirms the accessibility of custom topologies. The custom topology is created as in Figure 2.

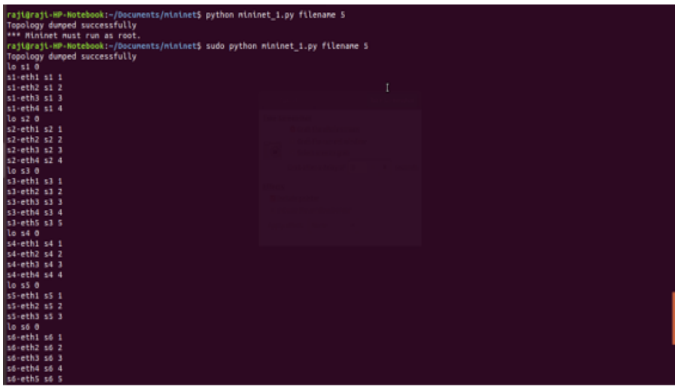

Figure.2: Topology Creation using Mininet

4.2. Controllers Design:

The dataset is generated for designing the SDN controllers that achieve malicious actions. The networks are administered by controllers that are arbitrarily chosen from the following controllers: i) Malicious controller, ii) Non-malicious controller. Using pyscript both compromised and non-compromised controllers are connected to the mininet as shown in the Figure 3.

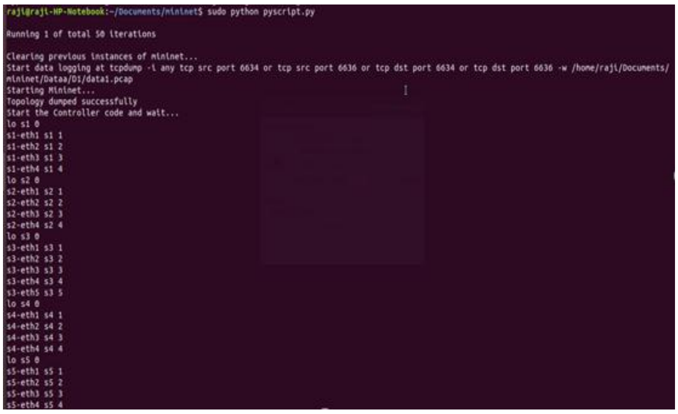

Figure.3: Connecting Controllers and Switch

4.3. Traffic Generation:

The traffic patterns are used for dataset generation. Ping-all traffic or randomized traffic pattern are generated only the mininet is being initialized. In Ping all traffic each host using utility command are used to interconnect with the other hosts.

Switches are run on TCP and UDP servers by using Iperf efficacy only in randomized traffic as in Figure. 4

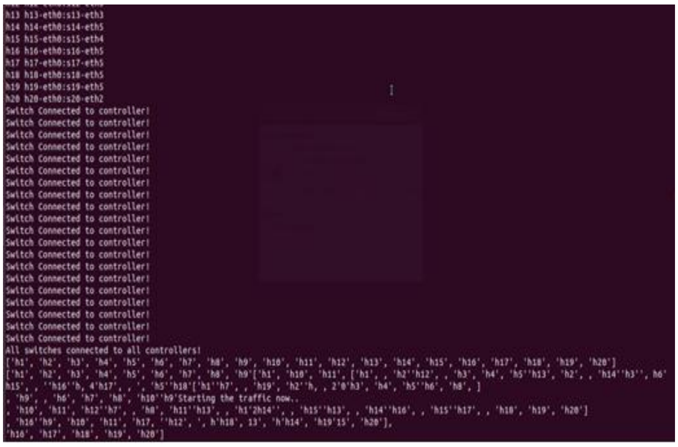

Figure.4: Traffic Generation

4.4. Creating Data Plane Packet Logs:

The generated dataset contains data plane packet logs are in Figure. 5. These logs are comprised by the both malicious and non-malicious controller. The dataset is created in the method of even number dataset are malicious and odd number dataset are non-malicious. The packets are captured in Wireshark as in Figure.5.

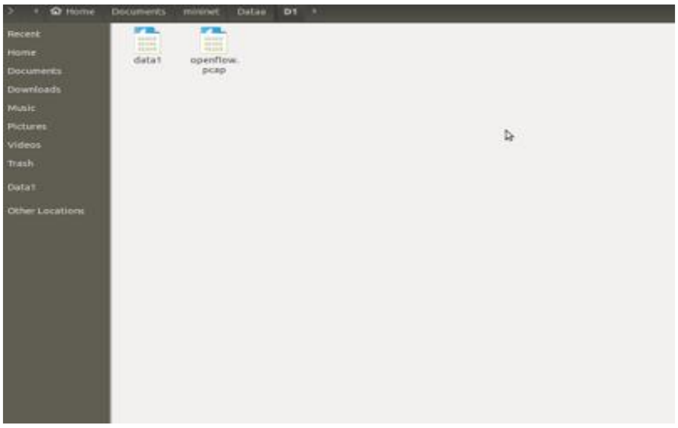

Figure 5: Generating Dataset

Wireshark is the network traffic analyzer and a vital tool for any security specialized or systems manager. It is permitted software to analyze network traffic in real time and also the finest tool for troubleshooting issues in network. All data packets are stored in the form of pcap file.

4.5. OpenFlow Feature Extraction:

A packet log contains different types of packets. All packets are captured through wireshark. From the pcap file, open flow information is only extracted by using pyshark. Figure.6 shows that the open flow packet flows are extracted.

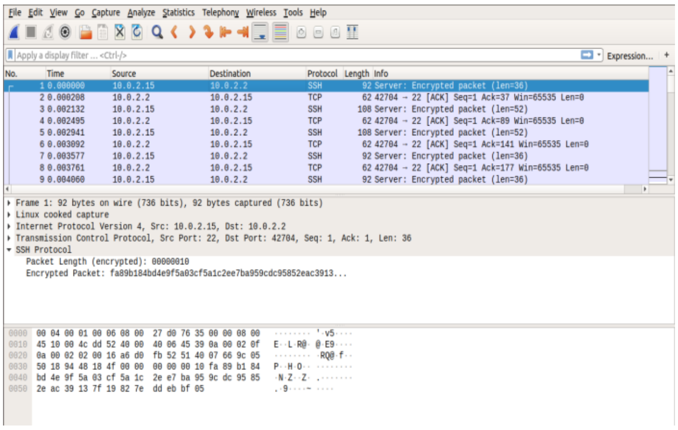

Figure.6: Data plane Packet Capture

The six features are extracted from the open flow packet flows. They are packet-in packet-out proportion, packet-in packet-out divergence, switch association index, precedence frequency spike index, variance of drop action, timeout frequency spike index is extracted from the open flow traces.

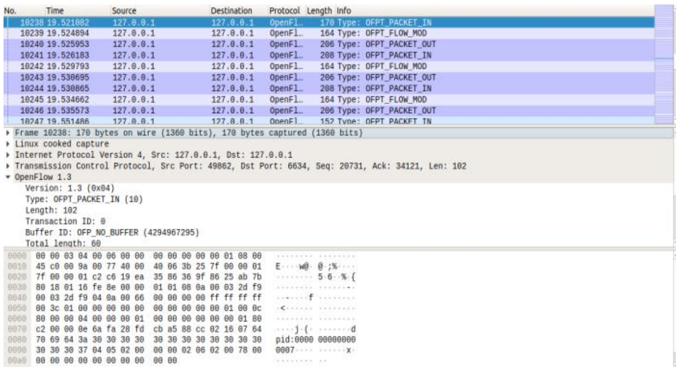

Figure.7: Open Flow Packet Extraction

4.6. Classification:

The above features are given into algorithms, which is machine learning concepts to categorize the network samples by benign or malicious. The random forest and the decision tree classifiers are used to categorize the network samples.

Figure.8: Open Flow Features Extraction Using Pyshark

The decision tree generates a tree over the take-out feature space and do not incline to specify over the data.

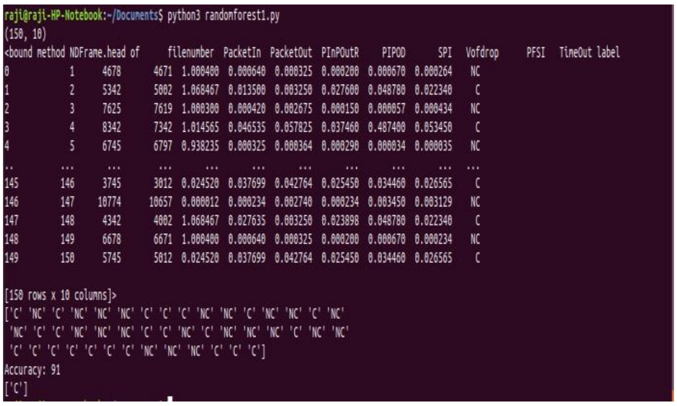

Figure.9: Random Forest Classifier output

Additionally, decision trees detect high correlation that occurs among numerous features and make the algorithm appropriate for the OpenFlow data traces.

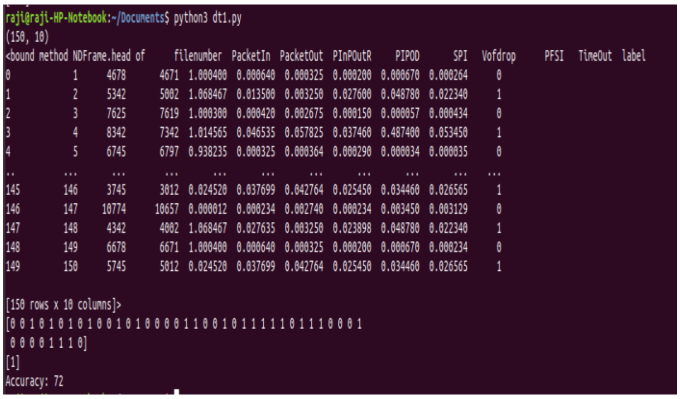

Figure.10: Decision Tree Classifier output

Decision tree gives 72 percent accuracy and RF gives 91 percent accuracy. Random forest shows the higher accuracy when related to decision tree. The results are shown in the Figure.9 and Figure.10.

5. CONCLUSION

The proposed system identifies the various types of threat vectors that characterize the malicious controllers in SDN. The presence of compromised controllers in control plane is detected using open flow traces. Any change in Open Flow protocol network performance has an equivalent change in the earlier switch and controller statement. Six SDN exact characteristics are filtered from Open Flow traces that are then provided to algorithms that are comes under machine learning concepts to identify the existence of malicious controller.

The random forest and decision tree classifiers used to classify the network samples. Random Forest classifier shows the higher accuracy. The additional features can improve the location and categorize the exact compromised controller and also find the nature of that attack. Furthermore, attacks will be generated, different strategy were planned to implement to detect the nature of controller.

About References

Wang projected a scheme *Sample Flow* (sFlow) [2] to ensure networks against DDoS attacks. sFlow captures the causes of DDoS, tags such flows, and subsequently drops the malicious flow packets. SDN architecture was recommended by Varadharajan et al. in [3] to implement security policies at switches through the addition of flow-rules. Such a policy-based method permits us to project security schemes based on dissimilar characteristics such as position, operator, host machine, and routing route. A prolonged modular architecture for controller design was planned by Polezhae et al. preserves the packets in order to diminish the cases of compromised controller [4]. An approach called AVANT-GUARD [5] was proposed by Yegneswaran et al. to stop DDoS resulting from SYN flood messages. A migration component was planned at Open Flow switches to reply TCP connection requests in its place of forwarding them to the destination node. It creates a connection, the switch forwards unaffected messages and drops out malicious counterparts.

References

[1] N. Anand, B. S. Sarath Babu, Manoj, "On detecting compromised controller in software defined networks," Computer Networks in the Journal of Elsevier, Vol. no. 137, pp. no. 107-118, 2018.

[2] Y. Lu, M. Wang, "An Easy Defense Mechanism against Botnet-based DDoS Flooding Attack Originated in SDN Environment Using sFlow," in: Proceedings of the 11th International Conference on Future Internet Technologies, CFI '16, ACM, New York, NY, USA, pp. 14–20,2016.

[3] K. Karmakar, V. Varadharajan, U. Tupakula, M. Hitchens, "Policy Based Security Architecture for Software Defined Networks," in: Proceedings of the 31st Annual ACM Symposium on Applied

Computing, SAC ’16, ACM, New York, NY, USA, pp. 658–663,2016.

[4] A. Shukhman, P. Polezhaev, Y. Ushakov, L. Legashev, V. Tarasov, N. Bakhareva, “Development of Network Security Tools for Enterprise Software-defined Networks,” in: Proceedings of the 8th International Conference on Security of Information and Networks, SIN ’15, ACM, New York, NY, US, pp. 224–228, 2015.

[5] S. Shin, V. Yegneswaran, P. Porras, G. Gu, “AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks,” in: Proceedings of the 2013 ACM SIGSAC Conference on Computer &’38; Communications Security, CCS ’13, ACM, New York, NY, USA, pp. 413–424, 2013.

[6] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, P. Porras, G. Gu, “Flow wars: systemizing the attack surface and defenses in software-defined networks,” IEEE/ACM Trans. Netw. 25 (6),3514–3530, 2017.

[7] J Systems and Networks Lab, OpenSDNDataset, 2017. https:// github.com/iist-sysnet/OpenSDNDataset.

[8] Tcpdump/libpcap public repository repository, 2017, http://www.tcpdump.org/, (Accessed on 07/26/2017).

[9] NLANR/DAST : Iperf - the TCP/UDP bandwidth measurement tool, 2005. http: //dast.nlanr.net/Projects/Iperf/ (November 2005).

[10] Open vSwitch, 2017, http://openvswitch.org/ , (Accessed on 07/25/2017)

[11] Ryu SDN framework, 2017, https://osrg.github.io/ryu/, (Accessed on 07/27/2017).

[12] NLANR/DAST: Iperf - the TCP/UDP bandwidth measurement tool, 2005. http: //dast.nlanr.net/Projects/Iperf/ (November 2005