

Intrusion Detection System with Snort

Biyang Fu
School of Computing
Clemson University
Clemson SC USA
biyangf@g.clemson.edu

Chenxi Hu
School of Computing
Clemson University
Clemson SC USA
chenxi@g.clemson.edu

ABSTRACT

Given the great amount of cases of network attacks and malicious activities, network security has drawn more attention in order to prevent and analyze this kind of problems at earlier stages. Various open-source projects have been established and aim at providing useful application development tools to facilitate problem solving. Snort is one of these projects with advanced functions and feasibility. In our project, we explore the intrusion detection system with Snort, establish the overall architecture and deploy the architecture into SDN environment. Finally, we observe the experimental results which verify the validity and feasibility of our project.

KEYWORDS

Intrusion Detection System, Snort, Open vSwitch, Ryu Controller

1 Introduction

Our project explores one of the network security problems called intrusion detection system based on Linux operating system. Given the potential attacks and threats, any system is supposed to monitor and alarm malicious activities. Consequently, various open-source projects are created to provide relevant functions, and one of them is called Snort. Snort is a prevalent network intrusion detection system with advanced scalability and portability.

In our project, we firstly install all components needed on our virtual machine. Then, we write customized rules for specific packet matching purpose. Furthermore, we implement dynamically rules installing which means we can install matching rules without rebooting the IDS. Lastly, we deploy Snort into the SDN environment and connect it with OpenFlow switch and Ryu controller, so that we can explore more functions in future. After conducting the experiment, the results show the feasibility of our ideas, and our exploration process exactly inspire us with more future work.

2 Background

IDS (Intrusion Detection System) is a device or application with capability of analyzing real-time data traffic and logging network data package activities. Snort IDS is an open source system with advanced scalability and portability. In Snort, the intrusion detection process is composed of several steps, for example, extracting features, defining rules, matching data packet with

predefined rules, affirming intrusion action. In our project, we follow the above process to implement our design.

Open vSwitch is another open source system used in our project which can provide software implementation of multilayer virtual switches in a virtual hardware environment. It can be easily installed on ubuntu system and configure customized network using ovs command. One characteristic application of Open vSwitch is constructing a software-based network within a virtual machine.

Ryu controller as an SDN controller is also included in our architecture. It is capable of providing network traffic handling and management with pre-defined application program interfaces (APIs). Typically, the role of SDN controller in a network topology is a brain of the whole system, which is responsible of transmitting information between switches and APIs to implement its functionality.

3 Motivations and Objectives

The continuous development based on the Internet and the wide application in people's life and work have made the development prospects of network information security technology considerable. And Snort IDS is a powerful network intrusion detection system. It has the capability of real-time data flow analysis and recording of IP network data packets. It can perform protocol analysis, match the content of network data packets, detect various attack methods, and provide real-time alarm to the attack. In IDS, if you need to change the matching rules, you usually need to stop the IDS network, then modify the rules, and reconfigure. However, in the process of restarting IDS, we lost the detection function of packets. Some malicious or unknown sources of packets may use this gap to escape the monitoring of the system. So, we also need to realize the function that change the rules without rebooting the IDS based on snort. Under the current general trend of Software-Defined Networking (SDN), we deploy IDS into the SDN architecture.

4 Design

The architecture of our project is shown in Figure 1. The whole architecture is built in an SDN environment. We install and configure Snort intrusion detection system and Ryu controller on our ubuntu virtual machine. In order to connect to all components used in our project, which include Ryu controller, Snort, Internet

and hosts, we create an OpenFlow switch by using Open vSwitch and deploy interfaces.

Firstly, the Snort module is based on the connection between host and server. We use Host A to ping Server shown in Figure 2.

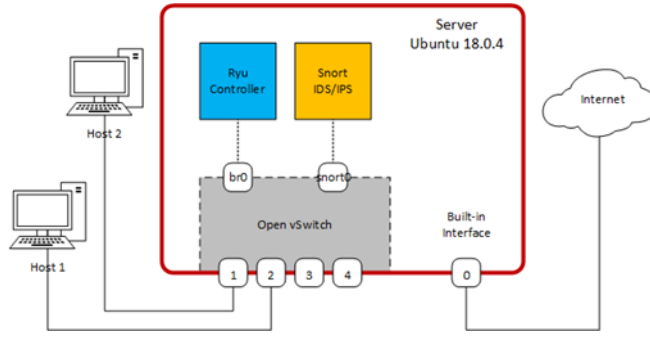


Figure 1: Project Architecture

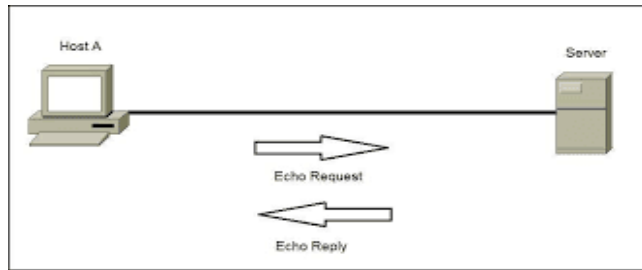


Figure 2: Ping Connection

When we ping IP address of Server from Host A, the ping command will first send an ICMP Echo Request to Server. After receiving it, Server will return an ICMP Echo Reply. If it does not return, it is timed out, and the specified network address will not be considered to exist.

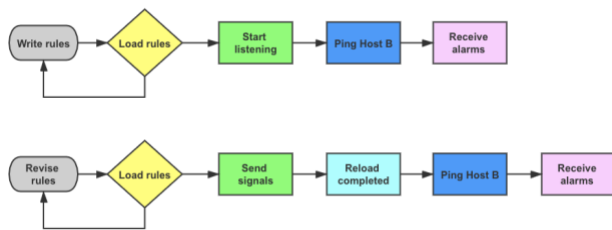


Figure 3: Flow Chart of Snort IDS

As shown in Figure 3, at first, we write the matching rule, and load it. If the load fails, you need to modify your rules to match snort's configuration. Next, we start listening. We cannot see anything now, because we haven't start connecting. So, we open a new terminal to ping IP address of Server. If it is successfully connected, we can receive the alarms based on the matching rules.

Next step, we should realize changing rules without rebooting IDS. We keep the snort working and open another terminal. We revise the rule and make sure it can be loaded successfully. Then, we send signals to tell snort that we have reloaded a new rule. Snort IDS will show the result of reload completed. We ping Server again and get the alarms according to our revised rules.

In the deployment module, we firstly install all components we need. Then, we add more network adapters on our virtual machine for necessary connections, and they are named as enp0s3, enp0s8, enp0s9, and enp0s10. Secondly, we create an OpenFlow Switch using Open vSwitch and add the four Ethernet ports to it. Then we connect the switch with hosts and Ryu controller to exchange OpenFlow packets and populate its flow tables. Figure 4 shows the network configuration.

```
chenxi@chenxi-VirtualBox: ~
File Edit View Search Terminal Help
chenxi@chenxi-VirtualBox:~$ sudo ovs-vsctl add-port br0 enp0s9 -- set interface enp0s9 ofport_request=3
chenxi@chenxi-VirtualBox:~$ sudo ovs-vsctl add-port br0 enp0s10 -- set interface enp0s10 ofport_request=4
chenxi@chenxi-VirtualBox:~$ sudo ovs-vsctl set-controller br0 tcp:10.0.10.10:6633
chenxi@chenxi-VirtualBox:~$ sudo ovs-vsctl show
52855d7a-ac08-4a84-af62-3d8cedb1a7ac
Bridge "br0"
  Controller "tcp:10.0.10.10:6633"
  fail_mode: standalone
  Port "br0"
    Interface "br0"
      type: internal
  Port "enp0s9"
    Interface "enp0s9"
  Port "enp0s8"
    Interface "enp0s8"
  Port "enp0s10"
    Interface "enp0s10"
  Port "enp0s3"
    Interface "enp0s3"
  ovs_version: "2.9.5"
chenxi@chenxi-VirtualBox:~$
```

Figure 4: Network Configuration

In the third step, we run Ryu controller which is a component-based software defined networking framework, and it supports various protocols for managing network devices. Also, we activate the Flow Manager application which provides a user-friendly interface to modify and monitor the flow tables in the network. Figure 5 shows that the controller connected to the switch successfully.

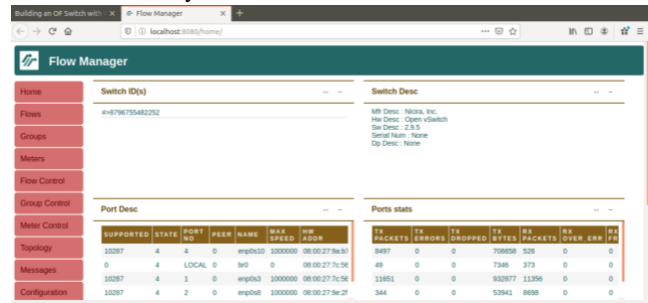


Figure 5: Flow Manager Home Webpage

In the fourth step, we integrate Snort into our environment. We create an interface connected to a switch port to receive packets for analysis. In the last step, we utilize a script provided by Ryu to link Snort IDS to Ryu.

5 Results

In the Snort module, the following is the rule we wrote at the beginning: alert icmp any any -> 192.168.56.101 any (msg: "ICMP PING "; sid:8886288). Among them, 192.168.56.101 is the IP address of the server. This rule means: Snort IDS will capture ICMP packets from any IP address port to any port of the server address and issue an alarm message. When we start listening and connecting to the server, IDS will issue the alert shown in Figure 6.

In fact, it shows that Snort captured ICMP Request packets sent from Host A to the server according to the rule. Next step, we modify the rules as follows: alert icmp any any -> 10.0.2.15 any (msg: "ICMP PING "; sid:8886288). We changed the server's address to the IP address of Host A, which is 10.0.2.15. And send a signal to tell Snort that the new rules have been loaded. Snort will display the following result indicating that the reload has been completed as Figure 7.

Then, we connect to the server again and we will get the result as shown in Figure 8. This shows that Snort has successfully captured the ICMP Reply packets sent from the server to Host A according to the modified rules.

```
pcap DAQ configured to passive.
Commencing packet processing
++ [0] enp0s3
04/14-00:54:26.766481 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 10.0.2.15 -> 192.168.56.101
04/14-00:54:27.789002 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 10.0.2.15 -> 192.168.56.101
04/14-00:54:28.812263 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 10.0.2.15 -> 192.168.56.101
04/14-00:54:29.836234 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 10.0.2.15 -> 192.168.56.101
04/14-00:54:30.860227 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 10.0.2.15 -> 192.168.56.101
04/14-00:54:31.884297 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 10.0.2.15 -> 192.168.56.101
```

Figure 6: Results of monitoring

In the deployment module, the eventual network configurations are shown in Figure 9. The ports are assigned to Ryu controller, Snort IDS, OpenFlow switch, hosts and Internet. Figure 10 shows the flow table on Flow Manager page. The Flow Manager shows that the flows added include additional OUTPUT action that copies packets to port 5 which is the Snort IDS port.

```
rule counts
  total rules loaded: 1
  duplicate rules: 1
  text rules: 1
  option chains: 1
  chain headers: 1
-----
port rule counts
      tcp      udp      icmp      ip
any      0      0      2      0
total    0      0      2      0
0 hosts loaded
.. swapping configuration
== reload complete
```

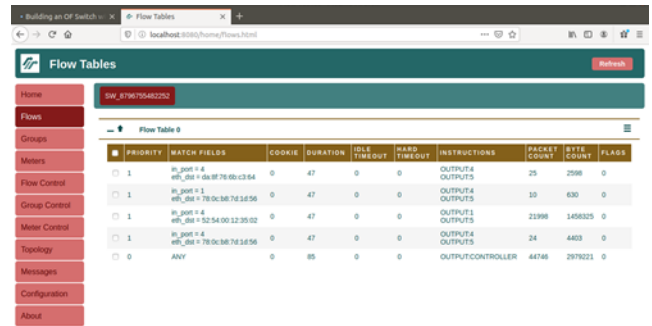
Figure 7: Reload completed

```
.. swapping configuration
== reload complete
04/14-01:10:45.360546 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 192.168.56.101 -> 10.0.2.15
04/14-01:10:46.380996 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 192.168.56.101 -> 10.0.2.15
04/14-01:10:47.404876 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 192.168.56.101 -> 10.0.2.15
04/14-01:10:48.428829 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 192.168.56.101 -> 10.0.2.15
04/14-01:10:49.452950 [**] [1:8886288:0] "ICMP PING" [**] [Priority: 0] [AppID
: ICMP] [ICMP] 192.168.56.101 -> 10.0.2.15
```

Figure 8: Monitoring results after modifying the rules

```
chenxi@chenxi-VirtualBox:~$ sudo ovs-vsctl show
52855d7a-ac08-4a84-af62-3d8cedb1a7ac
Bridge "br0"
  Controller "tcp:10.0.10.10:6633"
  fail_mode: standalone
  Port "enp0s3"
    Interface "enp0s3"
  Port "enp0s10"
    Interface "enp0s10"
  Port "enp0s9"
    Interface "enp0s9"
  Port "br0"
    Interface "br0"
    type: internal
  Port "enp0s8"
    Interface "enp0s8"
  Port "snort0"
    Interface "snort0"
    type: internal
  ovs_version: "2.9.5"
chenxi@chenxi-VirtualBox:~$
```

Figure 9: Network Ports



PRIORITY	WATCH FIELDS	COOKIE	DURATION	IDLE TIMEOUT	HARD TIMEOUT	INSTRUCTIONS	PACKET COUNT	BYTE COUNT	FLAGS
1	dl_port = 4 hl_dst = 4a:0f:78:00:13:64	0	47	0	0	OUTPUT4 OUTPUT5	25	2598	0
1	dl_port = 3 hl_dst = 78:0c:00:78:3d:56	0	47	0	0	OUTPUT4 OUTPUT5	10	630	0
1	dl_port = 4 hl_dst = 10:00:00:12:35:02	0	47	0	0	OUTPUT1 OUTPUT5	2398	1458325	0
1	dl_port = 4 hl_dst = 78:0c:00:78:3d:56	0	47	0	0	OUTPUT4 OUTPUT5	24	4403	0
0	ANY	0	85	0	0	OUTPUTCONTROLLER	44746	2879231	0

Figure 10: Flow Manager Flows Webpage

6 Conclusions and Future Work

In this project, we learn how Snort implements simple Deep Packet Inspection (DPI) based on matching rules. At the same time, Snort can also reload rules without restarting IDS. Note that only versions above Snort 2.9.9.x have this feature, and different versions display slightly different reload results. Deploying Snort's intrusion detection system into the SDN architecture will greatly expand our monitoring and control of network data traffic. Considering the situation of traffic overload, next we will study how to achieve traffic distribution in this architecture to reduce traffic congestion.

REFERENCES

- [1] Bremler-Barr, Anat, Yotam Harchol, David Hay, and Yaron Koral. "Deep packet inspection as a service." In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, pp. 271-282. 2014.
- [2] Dietrich, Noah. "Snort 2.9. 9. x on Ubuntu 14 and 16." Available: <https://www.snort.org/documents/snort-2-9-9-x-on-ubuntu-14-16>. [Último acceso: 13 Junio 2018] (2017).
- [3] Rehman, Rafeeq Ur. Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID. Prentice Hall Professional, 2003.