UNIVERSITY OF VIENNA

Summer Internship

Weekly report #2,3

Amir Sabzi
+989373107525
97.amirsabzi@gmail.com
August 2020

Letter of Submittal

To:	Prof. Stefan Schmid
From:	Amir Sabzi
Date:	August 25, 2020
Re:	Work Report: Report #2,3

Hi Professor,

In the last week, I didn't send you a report because I think it's better to finish Implementation and then write a report on the whole project. In this report, I discuss steps of implementation in detail, and also at the end, I come up with an idea to improve the covert channel and broaden its generality. I'll upload the codes and virtual machines in my Github in subsequent days. It would be great if you let me know your opinion about that.

Sincerely

Amir Sabzi

Table of Contents

1 Implementing a test environment **1**

2 Flow-reconfiguration in programmable data planes **3**

2.1 Threat model 3

2.2 Getting the flow tables locally 4

2.3 Execution of Attack scenario 4

3 Flow-reconfiguration: An idea **5**

3.1 Threat model 6

3.2 Execution of Attack scenario 7

List of Figures

1 Topology of the networks 2

2 Scenario-1 of the attack 3

3 An entry, dedicated to host *h1*, in flow table of *s1* 6

4 Scenario-2 of the attack 7

5 Timeline of attack procedure 8

1 Implementing a test environment

Although it's possible to install a Virtual machine form scratch and then build P4-behavior model and ONOS controller, I prefer to use the VM provided by ONOS community for the developers working on programmable data plane. To get the VM you should use following commands:

```
$ git clone <ONOS-Repository>
$ cd ./onos/tools/dev/p4vm
$ vagrant up dev
```

After installation of the VM with vagrant, I ssh into that. One possible problem in this step, which take a lot of time for me to overcome, is a error related to npm installation when you build the ONOS controller. To solve this problem it's crucial to modify the build script.

```
#Modification 1  onos GUI NPM install
#Add after $$NPM $$NPM? Args install of cmd
--registry=https://registry.npm.taobao.org
#Modification 2  onos GUI NPM build
#Add after "Chmod a + X. / node [modules/gulp/bin/" +
" export HTTPS_PROXY=http://ip:port &&" +
" export HTTP_PROXY=http://ip:port &&" +
```

I mentioned these modification because it is very time-consuming task to find solution of the problem and I hope it helps to readers who want to re-implement the test bed.

When the installation of VM is completed, I tried to make a a topology which use P4 switches in data plane. For this purpose I used A basic P4 code in VM. Forasmuch as the structure of the P4 code is independent of possibility of our attack, this cause no problem. You can see the topology which I selected for this part in the figure 1. To implement the topology

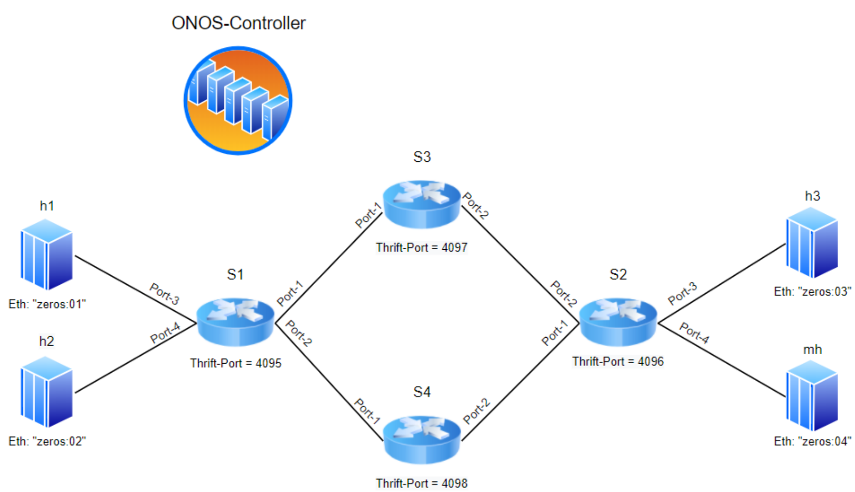

Figure 1: Topology of the networks

demonstrated in figure 1 I edited the code bmv2.py and add a topology class named *mytopo* to that. Now with the following command we can easily bring up our network.

```
sudo -E mn --custom $BMV2_MN_PY --switch onosbmv2 \
--controller remote --topo mytopo
```

2 Flow-reconfiguration in programmable data planes

2.1 Threat model

To develop a threat model, suppose we gain control over switches *s1* and *s2*. and these switches want to communicate with each other covertly. I supposed each switch can forge a *packet in* request for controller. This request will trigger a procedure which culminate in modification in the flow table of other switches.

Because in the mininet we can't generate packet in switches, I attached the host *mh* to the switch *s2* and I consider them as a unit. This scenario is depicted in the figure 2.

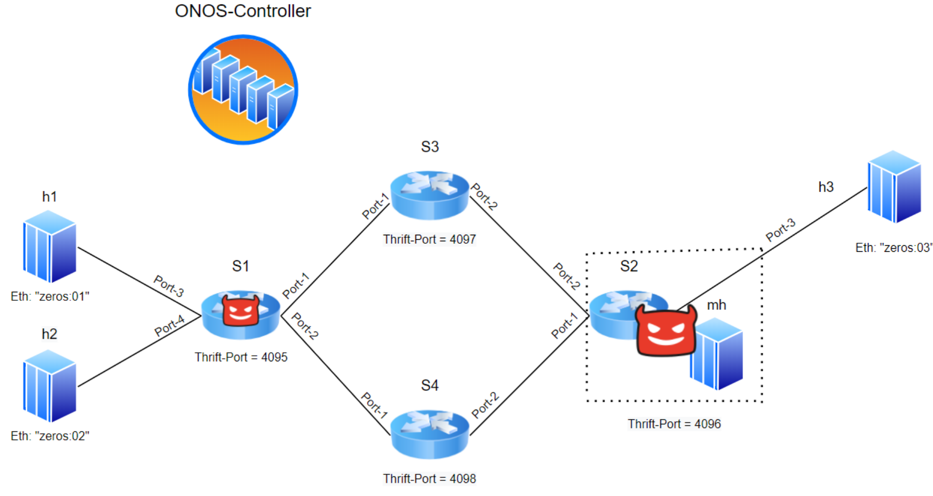

Figure 2: Scenario-1 of the attack

2.2 Getting the flow tables locally

In this part we need to have access to the flow tables of switches. Each P4 architecture model have its dedicated CLI. but I searched for CLI of the simple_switch which is the architecture I used in this part. To enable this local interface we should re-configure P4-behavior model with new options. To do that I re-configure and make/install the *bmv2* and *simple_switch_grpc* with the option, **- -with-thrift**. Now you can use command *simple_switch_CLI* with option –thrift-port and thrift port number of each switch, which set in topology class in the python code, to bring up a CLI for one of them. In the figure 1, you can see thrift port number of switches.

2.3 Execution of Attack scenario

Before building the topology, you should run the ONOS controller and install some features on this. For this purpose, use following commands in ONOS root directory:

```
$ bazel run onos-local -- clean debug
# In another terminal
$ export ONOS_APPS=drivers.bmv2,proxyarp,lldpprovider\
,hostprovider,fwd,mobility
$ bazel run onos-local -- clean
```

We'll use proxyarp applications for ping command. Also, the mobility application will help the network to keep track of the location of hosts. After bringing up the topology with mn command described in previous part, use *pingall* command in the mininet cli to set the flows. I it demonstrated in the figure 3-a, *s1* have flows including the flow to redirect packets to *h1*.
As you know there is no source authentication in Ethernet and IP protocols. So I developed a simple python script which use scapy to generate a packet with Ethernet address of *h1*, which is 00:00:00:00:00:01, as source address in host *mh*. After sending this packet, controller will add required flows on *s2* and remove installed flows dedicated to host *h1* from *s1*. thus by removing or Not removing these flows in a time interval we can implicitly transfer a single bit from *s2* to *s1*. You can see the entry of *s1* flow table before and after running python script in the figure 3.

3 Flow-reconfiguration: An idea

Although, I could easily automate the procedure described in the previous part, I think what if we haven't any control over switches!. Is it possible to implement flow reconfiguration attack in network?
To answer this question, we should develop another threat model. In the new model, we only control some hosts in the network. This scenario is

```
Dumping entry 0x4f000007
Match key:
* standard_metadata.ingress_port      : TERNARY   0001 &&& 01ff
* ethernet.src_addr                   : TERNARY   000000000003 &&& ffffffffffff
* ethernet.dst_addr                   : TERNARY   000000000001 &&& ffffffffffff
* ethernet.ether_type                 : TERNARY   0000 &&& 0000
* ipv4.src_addr                       : TERNARY   00000000 &&& 00000000
* ipv4.dst_addr                       : TERNARY   00000000 &&& 00000000
* ipv4.protocol                       : TERNARY   00 &&& 00
* scalars.local_metadata_t.l4_src_port: TERNARY   0000 &&& 0000
* scalars.local_metadata_t.l4_dst_port: TERNARY   0000 &&& 0000
Priority: 2147483636
Action entry: ingress.table0_control.set_egress_port - 03
```

(a) Before running the script

```
Dumping entry 0x3
Match key:
* standard_metadata.ingress_port      : TERNARY   0000 &&& 0000
* ethernet.src_addr                   : TERNARY   000000000000 &&& 000000000000
* ethernet.dst_addr                   : TERNARY   000000000000 &&& 000000000000
* ethernet.ether_type                 : TERNARY   0800 &&& ffff
* ipv4.src_addr                       : TERNARY   00000000 &&& 00000000
* ipv4.dst_addr                       : TERNARY   00000000 &&& 00000000
* ipv4.protocol                       : TERNARY   00 &&& 00
* scalars.local_metadata_t.l4_src_port: TERNARY   0000 &&& 0000
* scalars.local_metadata_t.l4_dst_port: TERNARY   0000 &&& 0000
Priority: 2147483641
Action entry: ingress.table0_control.send_to_cpu -
```

(b) After running the script

Figure 3: An entry, dedicated to host *h1*, in flow table of *s1*

much more realistic and easier to achieve. We want use flow reconfiguration technique to Implement a covert channel between two hosts which are physically disconnected.

3.1 Threat model

Suppose all switches in the network work properly. But we can control three hosts, which are one of them are physically disconnected from other switches and two other switch can communicate. This is a trivial scenario which is applicable in many situations. you can see this model in our topology in the figure 4. In this scenario host *mh* want to send information to the host *h2* implicitly. We suppose *s3* and *s4* will block all direct traffic from east to west or vice versa.

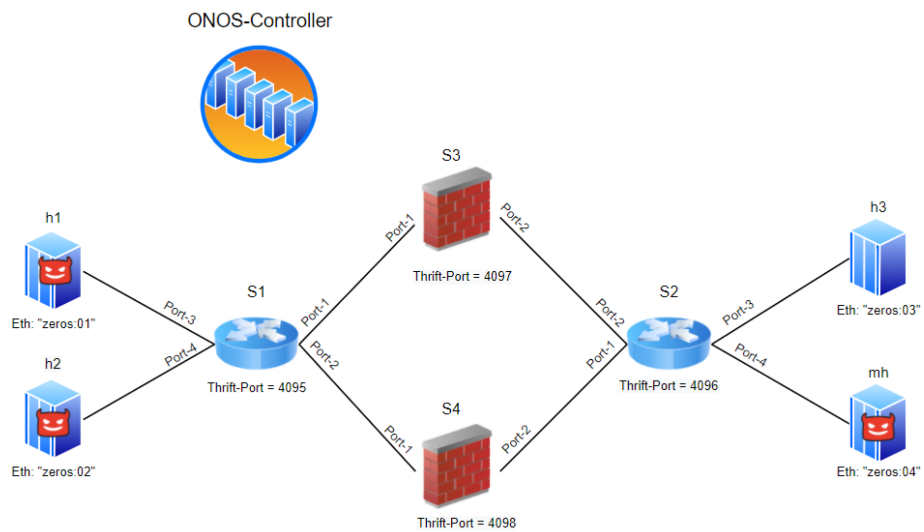

Figure 4: Scenario-2 of the attack

3.2 Execution of Attack scenario

As what we've done in the previous section, we can enforce the controller to remove flows dedicated to host *h1* in the switch *s1*. Although we don't have access to the switch flow tables from hosts, we can implicitly check existence/absence of some flows in the switches. For this purpose, I define a round which consists of three phases. In the first phase, host *mh* sends a bit. *mh* do this in such a way that if it wants to send 1, it will remove flows dedicated to *h1* in *s1* using the technique described in previous section. And to send a 0, *mh* simply do nothing in this phase. Then in the phase two, *h2* try to ping *h1*. If *mh* in the previous phase had removed the flows, *h2* would NOT be able to ping *h1*. But if *h2* can ping *h1*, this means *mh* didn't remove the flows in phase 1. And Finally, in the phase 3, *h1* ping *h2* to install appropriate flows to achieve connectivity with *h2*. The timeline of these three phase is demonstrated in the figure 5.

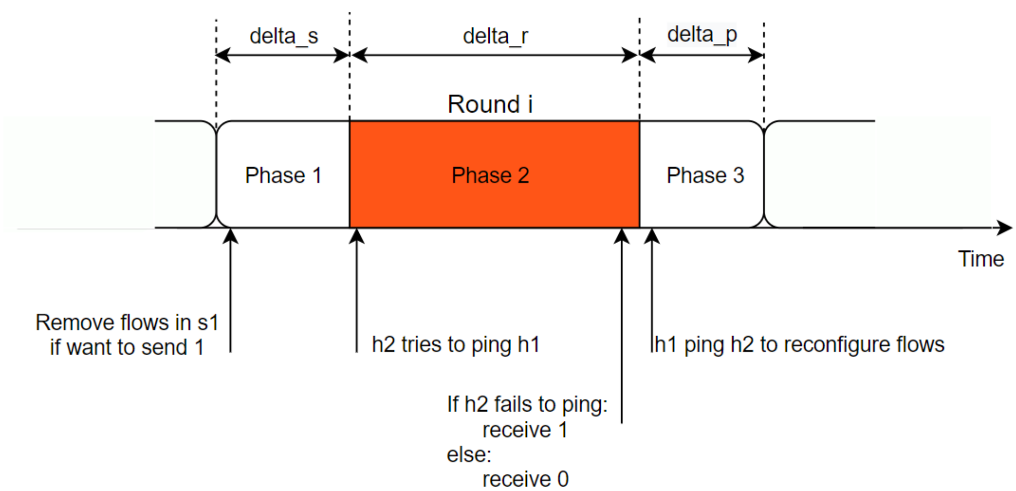

Figure 5: Timeline of attack procedure

To implement the covert channel, I wrote three python scripts. First, *sender.py* which is executed in *mh* and is responsible to send bit stream. Second, *receiver_h2.py* which is executed in *h2* and receive bit stream in phase 2. And the last, *receiver_h1.py* which is executed in *h1* and works as heart beat for system and re-configure flows on *s1* by pinging *h2*.

I test these code and I succeeded to transmit a relatively long bit stream with small error. From this link, you can see an execution of the covert channel. Currently, I set the variables delta_r, delta_s and delta_p equal to 0.3. So almost we can transmit one bit per second. But if you have any idea to improve this method, I will appreciate that.

sincerely

Amir Sabzi