Building IDS based on NFV

8570 Final Project Report - Team 9

Seth Li, Dhruv Jain, Yang Guo, Anukul Raj

School of Computing, Clemson University

Wednesday May 3rd, 2017

***Abstract*—Over several years, middleboxes have become a fundamental part of the network today. Although there are lots of advantages of middleboxes, but it also comes with a lot of problems due to the fact that they are hardware-based. So it has some bad influences like costly, difficult to manage and their functionality is hard to change. Launching a new service is difficult and takes too long. This is solved by NFV, we try to define a new NFV in clickOS to build the virtual intrusion detection system. For this whole project, we used tools like, clickOS, NFV, DPI, Xen. etc**

Keywords—intrusion detection system, network function virtualization, software defined networking, ClickOS.

I. Introduction

The goal of this project is to develop a virtual intrusion detection system (vIDS) based on NFV in ClickOS. There are problems with traditional network middleboxes like they are costly, difficult to manage, functionality is hard to change and launching a new service is difficult and takes too long. The solution to this problem is to virtualize the middlebox. In this project we implement a virtualized middlebox deep packet inspection which is an integral part of many middleboxes. We will look at the architecture in section II, setup and environment that we used in the project in section III and finally see the results of the experiments in section IV.

II. Architecture

The figure below shows the high level architecture of the clickos setup and the project. The hardware is the osboxes virtual machine on which xen is installed. OpenvSwitch is the de facto virtual switch for XEN which is running the ClickOS instances on top of it. The setup is mentioned in section 3.

A. Intrusion Detection System

Intrusion detection provides the following: Monitoring and analysis of user and system activity, auditing of system configurations and vulnerabilities, assessing the integrity of critical system and data files, statistical analysis of activity patterns based on the matching to known attacks, abnormal activity analysis and operating system audit.

In this project we will be building a Signature-based IDS which looks of specific patterns and known malicious instructions.

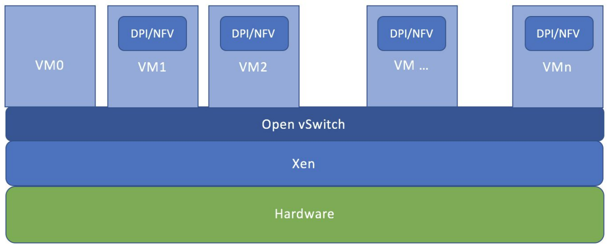


Fig. 1. Architecture level view of the clickos testbed

B. XEN

Xen project is a popular hypervisor which allows to run multiple operating systems to execute on the same computer hardware concurrently [1].

In this project we used Xen to run multiple instances of click operating system.

C. ClickOS

ClickOS is a high performance, XEN based virtualized platform optimized for middlebox processing. ClickOS virtual machines are very small, boot quickly have very little delays and biggest advantage is over thousands of them can run parallely. It implements an extensive overhaul of XEN's input output subsystems, including changes to the backend switch, virtual net device and back & front end drivers [2].

It is based on Click, which is software architecture for building flexible and configurable router. Click configurations are modular and easy to extend. A click router is assembled from packet processing modules called elements. Individual elements implement simple router functionalities such as, packet classification, queuing, scheduling and interfacing with network devices [3].

MiniOS is a tiny operating system kernel which comes with the XEN hypervisor sources. Each ClickOS virtual machine consists of Click modular router running on top of MiniOS as shown in Figure 2.

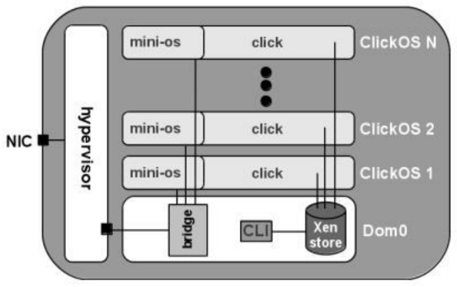


Fig. 2. ClickOS Architecture - The system consists of a set of ClickOS virtual machines (vms), each composed of Click element running on top of mini-os.

D. *Open vSwitch*

Open vSwitch is software implementation of a multilayer network switch. It was designed in order to introduce automation in the network while still being able to support the standard interfaces and protocols. The main advantage of using Open vSwitch is that it can be operated both as software-based network switch running inside a virtual machine hypervisor as well as the control stack for dedicated hardware switching. It is the default network switch in the XenServer virtualization platform [5].

We used version 2.5.0 for this project. ovs-vsctl is a utility for querying and updating the configuration of ovs-vswitchd.

In this project ovs sits on top of Xen hypervisor and acts as an SDN controller which allows to create multiple instances of clickos.

E. *Deep Packet Inspection*

Deep packet inspection is a form of computer network packet filtering that examines the data or header part of a packet as it passes an inspection point, searching for protocol non compliance, viruses, spam, intrusions, or defined criteria to decide whether the packet may pass or if it needs to be routed to a different destination for the purpose of collecting statistical information that functions at the application layer of the OSI [6].

It combines the functionality of an IDS (Intrusion Detection System) and IPS (Intrusion Prevention System) with a traditional stateful firewall. The benefit of this approach is that neither the IDS/IPS or the stateful firewall is capable of detecting certain attacks which are not possible based on individual components [6].

DPI is generally used to prevent attacks from viruses and worms at wire speeds. They are also very effective against buffer overflow attack, denial of services (DoS) attacks, sophisticated intrusions, and a small percentage of worms that fit within a single packet [6].

F. *Scapy*

Scapy is a powerful interactive packet manipulation program that helps user to send, sniff and dissect & forge network packets. It also sends packets on the wire, capture them and match the request and replies [7].

Scapy also supports actions such as addition of some new protocols or new functions by extending its source files. This feature makes scapy very useful for design and experimental purposes.

It is also used for tasks like scanning, tracerouting, probing, unit tests, attacks and network discovery [7].

G. *Floodlight*

Floodlight is an SDN controller that works with the OpenFlow protocol to to orchestrate traffic flows in a software-defined environment. It defines the open communication protocols in an SDN environment enabling the SDN controller to speak to forwarding planes (routers, switches, etc) in order to make changes in the network [8].

The benefits of using the floodlight controller is that it facilitates the users with the ability to easily adapt software and develop applications. It has representational state transfer application program interfaces (REST APIs) that makes it easier for the user to program interface with the product [8].

In our project we used the traffic monitor software floodlight version 1.2 Floodlight monitors the clickos traffic. It detects the network packet flow rate and perform actions such as create a new clickos instance dynamically.

III. Goals

The goal of this project is to develop a virtual intrusion detection system (vIDS) based on NFV in ClickOS.

First we need to be familiar with how ClickOS works to build virtual network functions.

Basic required functions in this vIDS:

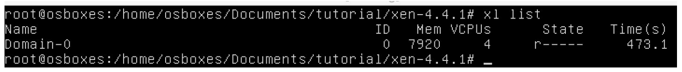
- Support Deep Packet Inspection by string match.
- Can install/delete signatures in the IDS dynamically.
- Can create a new vIDS instance (probably a new ClickOS instance) when the existing vIDS is overloaded.

IV. Setup

In this section we will walk through the different steps we took to setup our environment for the project. The initial step was to setup a virtual machine (VM) that runs ClickOS, and then go on to create our intrusion detection system.

A. *ClickOS setup*

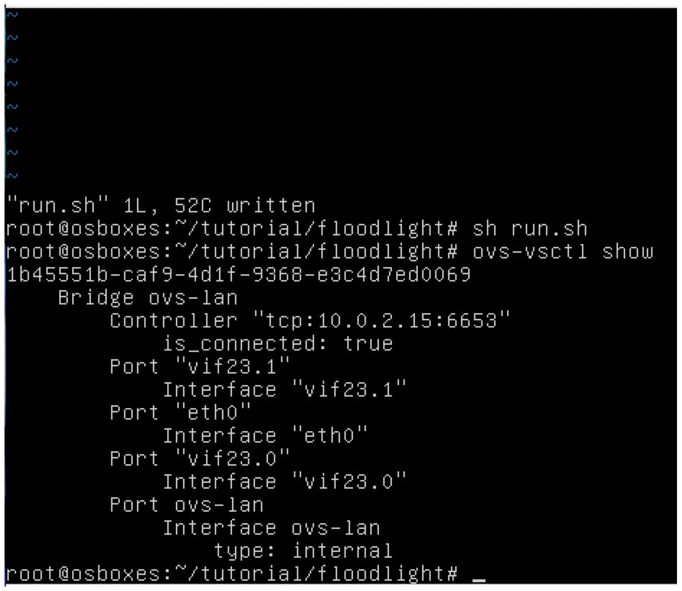
- Install Xen [4] - We installed Xen version 4.4.1
- We installed all the softwares and tools under the directory called tutorial.
- To verify Xen has been installed successfully we ran the command ‘xl list’. Domain0 is the service console, the initial domain started by the Xen hypervisor on boot.



```
root@osboxes:/home/osboxes/Documents/tutorial/xen-4.4.1# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0  7920     4     r-----     473.1
root@osboxes:/home/osboxes/Documents/tutorial/xen-4.4.1# _
```

Fig. 3. Output from ‘xl list’ command showing that Xen has been successfully installed on the VM.

- Install OpenvSwitch

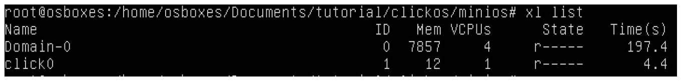


```
~
~
~
~
~
~
~
~
"run.sh" 1L, 52C written
root@osboxes:~/tutorial/floodlight# sh run.sh
root@osboxes:~/tutorial/floodlight# ovs-vsctl show
1b45551b-caf9-4d1f-9368-e3c4d7ed0069
    Bridge ovs-lan
        Controller "tcp:10.0.2.15:6653"
            is_connected: true
        Port "vif23.1"
            Interface "vif23.1"
        Port "eth0"
            Interface "eth0"
        Port "vif23.0"
            Interface "vif23.0"
        Port ovs-lan
            Interface ovs-lan
                type: internal
root@osboxes:~/tutorial/floodlight# _
```

Fig. 4. Output from ‘ovs-vsctl show’ command showing that OpenvSwitch is running correctly on the VM.

- ClickOS Instance

We made a script called createinstance.sh to destroy and create a new click instance.



```
root@osboxes:/home/osboxes/Documents/tutorial/clickos/minios# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0  7857     4     r-----     197.4
click0                                       1    12     1     r-----       4.4
```

Fig. 5. Output from ‘xl list’ command showing that clickos instance is running.

- ClickOS Element

Clickos element is the main part of the project. We created a new element that is a intrusion detection system using deep packet inspection. The click element is placed in the directory /clickos/elements/standard named ids.cc and ids.hh. We were able to successfully compile and run ids element.

- ClickOS Configuration

We created a new configuration file in click under the directory /clickos/minios named ids.click to run the new click element we just created. The clickos configuration reads packets from network device, perform deep packet inspection and sends packets to network device. The click configuration file also filters IP address directing the traffic to click element and print the output using print function.

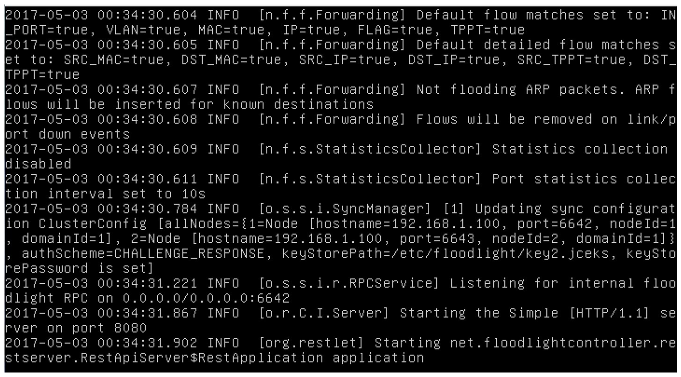
- Sending packets

We need to construct packets and send them to ClickOS. We used a python library called scapy to send packets to click. The python version running on the VM was 2.7.

- Traffic Monitoring and Connection between OpenvSwitch and Floodlight

We used floodlight open SDN controller to monitor traffic. After downloading, build the floodlight. The screenshot in figure 6 shows floodlight running successfully.

We needed to connect the openvSwitch to floodlight. Combine ovs bridge to eth2 and configure the IP address of ovs. Find the DPID for ovs in the Floodlight console.

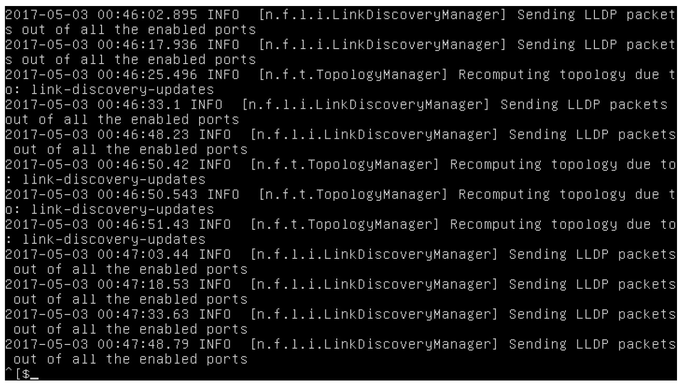


```
2017-05-03 00:34:30.604 INFO  [n.f.f.Forwarding] Default flow matches set to: IN
_PORT=true, VLAN=true, MAC=true, IP=true, FLAG=true, TPPT=true
2017-05-03 00:34:30.605 INFO  [n.f.f.Forwarding] Default detailed flow matches s
et to: SRC_MAC=true, DST_MAC=true, SRC_IP=true, DST_IP=true, SRC_TPPT=true, DST_
TPPT=true
2017-05-03 00:34:30.607 INFO  [n.f.f.Forwarding] Not flooding ARP packets. ARP f
lows will be inserted for known destinations
2017-05-03 00:34:30.608 INFO  [n.f.f.Forwarding] Flows will be removed on link/p
ort down events
2017-05-03 00:34:30.609 INFO  [n.f.s.StatisticsCollector] Statistics collection 
disabled
2017-05-03 00:34:30.611 INFO  [n.f.s.StatisticsCollector] Port statistics collec
tion interval set to 10s
2017-05-03 00:34:30.784 INFO  [o.s.s.i.SyncManager] [1] Updating sync configurat
ion ClusterConfig [allNodes={1=Node [hostname=192.168.1.100, port=6642, nodeId=1
, domainId=1], 2=Node [hostname=192.168.1.100, port=6643, nodeId=2, domainId=1]}
, authScheme=CHALLENGE_RESPONSE, keyStorePath=/etc/floodlight/key2.jceks, keySto
rePassword is set]
2017-05-03 00:34:31.221 INFO  [o.s.s.i.r.RPCService] Listening for internal floo
dlight RPC on 0.0.0.0/0.0.0.0:6642
2017-05-03 00:34:31.867 INFO  [o.r.C.I.Server] Starting the Simple [HTTP/1.1] se
rver on port 8080
2017-05-03 00:34:31.902 INFO  [org.restlet] Starting net.floodlightcontroller.re
stserver.RestApiServer$RestApplication application
```

Fig. 6. Output from floodlight console

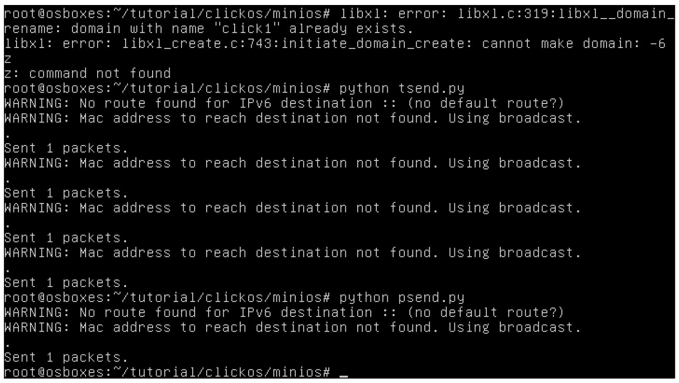
- Monitor python script with Floodlight REST API

We wrote a python script mon.py to monitor traffic. Running the monitor python script - creating clickos instances dynamically when the number of packets increase to 50.



```
2017-05-03 00:46:02.895 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packet
s out of all the enabled ports
2017-05-03 00:46:17.936 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packet
s out of all the enabled ports
2017-05-03 00:46:25.496 INFO  [n.f.t.TopologyManager] Recomputing topology due t
o: link-discovery-updates
2017-05-03 00:46:33.1 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets 
out of all the enabled ports
2017-05-03 00:46:48.23 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets
 out of all the enabled ports
2017-05-03 00:46:50.42 INFO  [n.f.t.TopologyManager] Recomputing topology due to
: link-discovery-updates
2017-05-03 00:46:50.543 INFO  [n.f.t.TopologyManager] Recomputing topology due t
o: link-discovery-updates
2017-05-03 00:46:51.43 INFO  [n.f.t.TopologyManager] Recomputing topology due to
: link-discovery-updates
2017-05-03 00:47:03.44 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets
 out of all the enabled ports
2017-05-03 00:47:18.53 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets
 out of all the enabled ports
2017-05-03 00:47:33.63 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets
 out of all the enabled ports
2017-05-03 00:47:48.79 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets
 out of all the enabled ports
^[$_
```

Fig. 7. Output from floodlight console



```
root@osboxes:~/tutorial/clickos/minios# libxl: error: libxl.c:319:libxl__domain_
rename: domain with name "click1" already exists.
libxl: error: libxl_create.c:743:initiate_domain_create: cannot make domain: -6
z
z: command not found
root@osboxes:~/tutorial/clickos/minios# python tsend.py
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
root@osboxes:~/tutorial/clickos/minios# python psend.py
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
root@osboxes:~/tutorial/clickos/minios# _
```

Fig. 8. Running the tsend script

V. Deep Packet Inspection Algorithm

Deep Packet Inspection is an important part of any Intrusion Detection System. In this a packet as it passes through an inspection point is analyzed for intrusion.

For our project as part of deep packet inspection, we tried to find signatures in the packet resembling to an already present signature and based on the result deciding whether a packet should be allowed to proceed in the network or should it be treated as an intrusion. Therefore, pattern matching became an important step.

In order to implement the pattern matching in our project, we decided to use the Knuth-Morris-Pratt or the KMP algorithm as it is popularly known. The KMP algorithm is a popular pattern searching algorithm which searches for the presence of a signature or pattern or word in any text. The benefits of using KMP algorithm in our program is that it improves the performance of the program.

The KMP pattern matching algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in pattern) of the pattern and improves the worst case complexity to O(n). The concept behind this is that we preprocess the pattern (signature) that we want to look for so that when there is a partial match between the pattern and original data, we already know some of the text in the next search window. Thus, we can skip over those bits and need not to compare it again and again.

Now lets us discuss how the algorithm actually works, suppose we are trying to find the pattern *"abababca"* in the data *"bacbababaabcbab"*. Now the key to KMP algorithm is that we preprocess the pattern *"abababca"* and prepare a partial match table, to know how many bits to skip in the next search window when we found a mismatch after a partial match in the data "*bacbababaabcbab*". This is achieved by matching the longest proper prefix to the suffix or vice versa and counting the maximum length of that string and keeping that value in the partial table.

Before, proceeding with the example let us explain first what we mean by proper prefix or suffix, the proper prefix of any string is a list of all the prefix or suffix of the string where the whole string is not allowed.

Now let us examine the pattern we are looking "*ababca*".

Pattern =>	a b a b a b c a
Bits, i =>	0 1 2 3 4 5 6 7
Partial Table =>	0 0 1 2 3 4 0 1

Let us look at the from the second bit *(i=1)*, we are not looking at *i=0*, as it signifies only one character so the first value in the partial table is always zero.

Iteration 2 (i =1):

Bits => ab

Proper Prefix=> a

Proper Suffix=> b

So, here there is no match so the value in Partial Table is 0.

Iteration 3 (i =2):

Bits=> aba

Proper Prefix=> a, ab

Proper Suffix=> a, ba

So, here we have a match "a", its length is 1. So the value in the partial table will be 1.

Iteration 4 (i =3):

Bits=> abab

Proper Prefix=> a, ab, aba

Proper Suffix=> b, ab, bab

So, here we have a match "ab", its length is 2. So the value in the partial table will be 2.

Iteration 5 (i =4):

Bits=> ababa

Proper Prefix=> a, ab, aba, abab

Proper Suffix=> a, ba, aba, baba

So, here we have two matches match "a" and "aba", but we choose the longest one. So the length of string "aba" is 3. Hence the value in the partial table will be 3.

Iteration 6 (i =5):

Bits=> ababab

Proper Prefix=> a, ab, aba, abab, ababa

Proper Suffix=> b, ab, bab, abab, babab

So, here we have again have two matches, "ab" and "abab". We choose the longest one and hence the value in the partial table will be 4.

Iteration 7 (i =6):

Bits=> abababc

Proper Prefix=> a, ab, aba, abab, ababa, ababab

Proper Suffix=> c, bc, abc, babc, ababc, bababc

As here, we have no match so the value in the partial table will be 0.

Iteration 8 (i =7):

Bits=> abababca

Proper Prefix=> a, ab, aba, abab, ababa, ababab, abababc

Proper Suffix=> a, ca, bca, abca, babca, ababca, bababca

So, here we only have one match "a", hence the value in the partial table will be 1.

This is how the partial table is generated. Now let us see how it is actually used in Pattern Matching.

As mentioned above, we are finding the pattern *"abababca"* in the data *"bacbababaabcbab"*.

Data (Txt) => bacbababaabcbab

Pattern (Ptn)=> abababca

So we begin the comparison by converting both strings into character array and matching first bit of the data to the first bit of the pattern and if there is a match, we match the second bit of the data to the second bit of the pattern and so on till we find a mismatch or a complete match. In case if we find a partial match of length n, is found then to know the number of bits that are to be skipped are determined by using the values in the partial table. If the value in the partial table m, corresponding the partial match length subtracted by 1 (n-1), is greater than 1 then the bits skipped is calculated by partial match length subtracted by value in table for the bit partial length subtracted by 1, that is, n - m->[n-1].

Iteration 2 (i =1):

```
Txt=>    b a c b a b a b a a b c b  a b
Ptn=>      a b a b a b c a
```

Here, we find mismatch at the second bit only. Looking up the partial table we know the value for first bit in pattern in 0. The value at partial m1atch table for the bit m->[n-1] is that is bit 0 is 0. So we do not skip any bit.

The next match we have is at Iteration 5.

Iteration 5 (i =4):

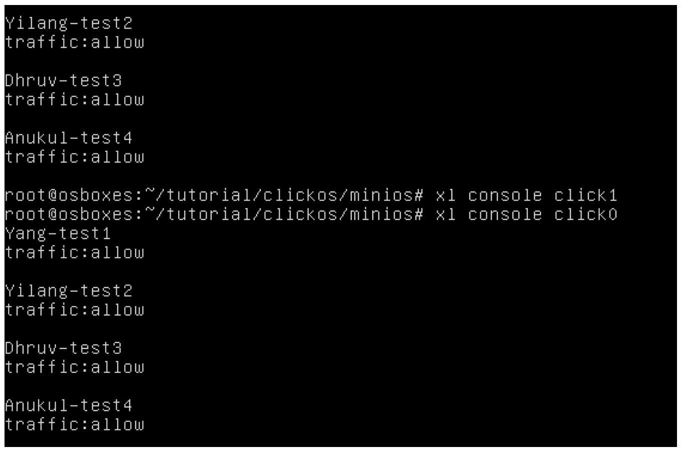
```
Txt=>    b a c b a b a b a a b c b  a b
Ptn=>            a b a b a b c a
```

Here we have a partial match length, n = 5 *(ababa)*, the value in table for bit m->[n-1], that is m[4] is 3. So the number of bits skipped will be n - m->[n-1], that is 5-3 = 2.

So, in the next iteration instead starting the comparison form the next window we start by skipping the first two bit as they have already been matched and so on [9].

VI. Results

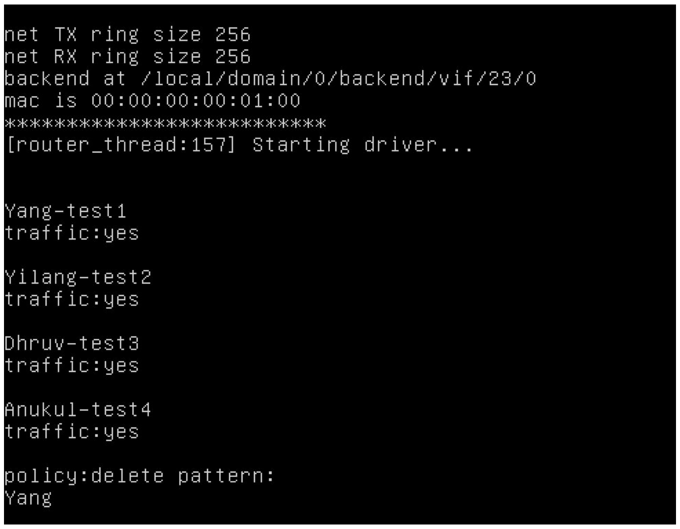
In this project we were able to successfully create a click element intrusion detection system. The config.xen is the xen configuration file that is present in the repository, this is how we create new click instance. tsend sends traffic - 'python tsend.py'. psend.py policy script where we are deleting yang pattern. This is where we edit to modify add and delete patterns.



```
Yilang-test2
traffic:allow

Dhruv-test3
traffic:allow

Anukul-test4
traffic:allow

root@osboxes:~/tutorial/clickos/minios# xl console click1
root@osboxes:~/tutorial/clickos/minios# xl console click0
Yang-test1
traffic:allow

Yilang-test2
traffic:allow

Dhruv-test3
traffic:allow

Anukul-test4
traffic:allow
```

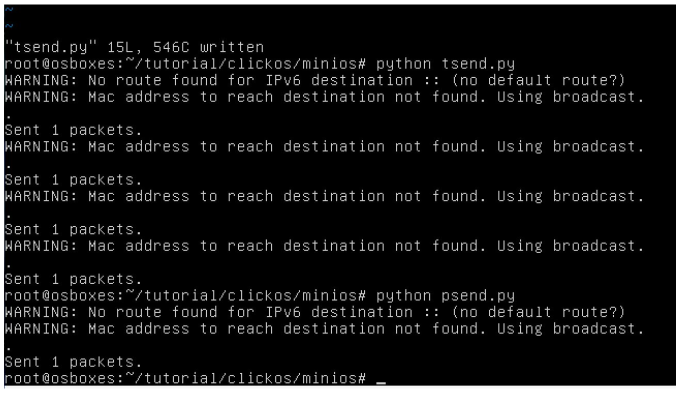
Fig. 9. The above screenshot shows the output from 'xl console click0' command.

The above screenshot in fig. 9 shows that packets containing our names were allowed inside the network because they matched the pattern from the rules. Any packet not containing our team member names will be dropped because it will not match all or any part of the IP payload.



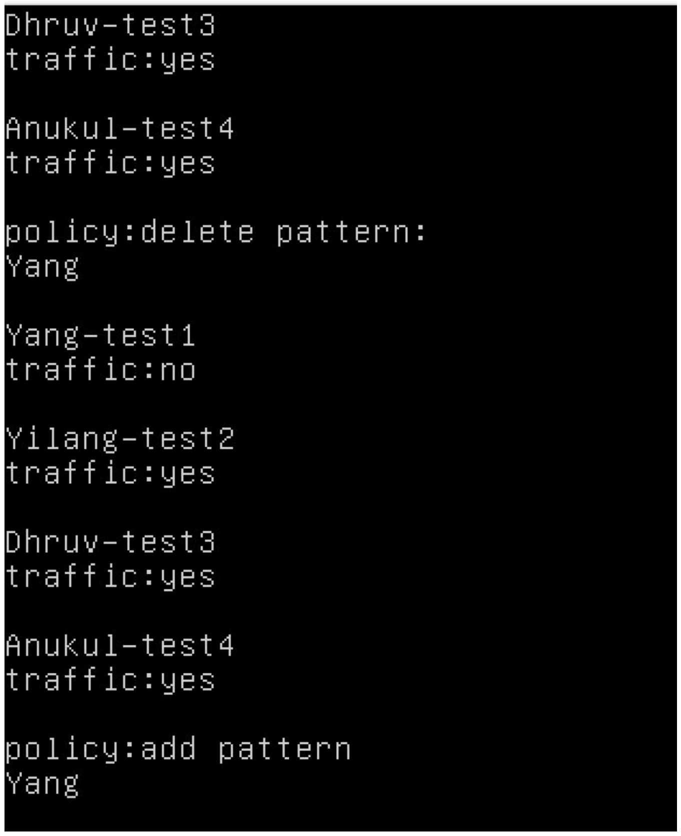
```
net TX ring size 256
net RX ring size 256
backend at /local/domain/0/backend/vif/23/0
mac is 00:00:00:00:01:00
**************************
[router_thread:157] Starting driver...


Yang-test1
traffic:yes

Yilang-test2
traffic:yes

Dhruv-test3
traffic:yes

Anukul-test4
traffic:yes

policy:delete pattern:
Yang
```

Fig. 10. The above screenshot shows the output from ‘xl console click0’ command. We deleted the Yang pattern from the policy



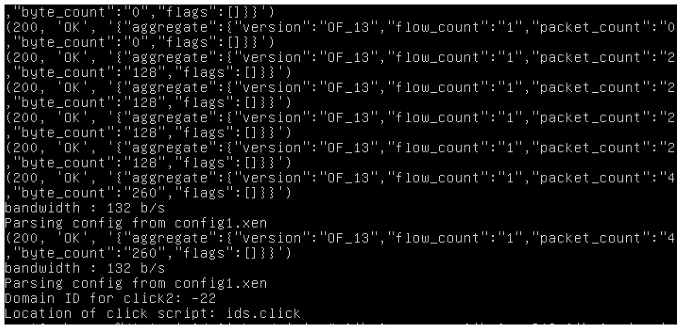
```
~
~
"tsend.py" 15L, 546C written
root@osboxes:~/tutorial/clickos/minios# python tsend.py
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
root@osboxes:~/tutorial/clickos/minios# python psend.py
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
root@osboxes:~/tutorial/clickos/minios# _
```

Fig. 11. The above screenshot shows the output after running the tsend.py script which send the packets to click.



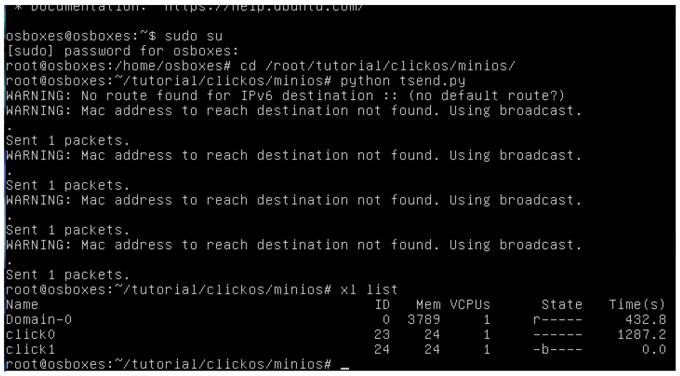
```
Dhruv-test3
traffic:yes

Anukul-test4
traffic:yes

policy:delete pattern:
Yang

Yang-test1
traffic:no

Yilang-test2
traffic:yes

Dhruv-test3
traffic:yes

Anukul-test4
traffic:yes

policy:add pattern
Yang
```

Fig. 12. The above screenshot shows the output from ‘xl console click0’ command. We added the pattern back of Yang.



```
,"byte_count":"0","flags":[]}}')
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"0
,"byte_count":"0","flags":[]}}')
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"2
,"byte_count":"128","flags":[]}}')
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"2
,"byte_count":"128","flags":[]}}')
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"2
,"byte_count":"128","flags":[]}}')
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"2
,"byte_count":"128","flags":[]}}')
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"4
,"byte_count":"260","flags":[]}}')
bandwidth : 132 b/s
Parsing config from config1.xen
(200, 'OK', '{"aggregate":{"version":"OF_13","flow_count":"1","packet_count":"4
,"byte_count":"260","flags":[]}}')
bandwidth : 132 b/s
Parsing config from config1.xen
Domain ID for click2: -22
Location of click script: ids.click
```

Fig. 13. The above screenshot shows the output from running the mon.py script.



```
 * Documentation:  https://help.ubuntu.com/

osboxes@osboxes:~$ sudo su
[sudo] password for osboxes:
root@osboxes:/home/osboxes# cd /root/tutorial/clickos/minios/
root@osboxes:~/tutorial/clickos/minios# python tsend.py
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
root@osboxes:~/tutorial/clickos/minios# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0  3789     1     r-----     432.8
click0                                      23    24     1     ------    1287.2
click1                                      24    24     1     -b----       0.0
root@osboxes:~/tutorial/clickos/minios# _
```

Fig. 14. The above screenshot shows the output from ‘xl list’ command. You can see a new click instance called click1 which is created dynamically when the bandwidth is more than 50.

VII. Conclusion and Future Work

Deep Packet Inspection is a common service used by today’s middleboxes. DPI is a scalable option which means it can easily be exported as a stand-alone network service. There is some future work which could be potentially done on our project. Some of the tasks like payload processing - decryption and decompression can be implemented. Session reconstruction or connection tracking can be implemented for sessions processing rather than packet processing. Header and protocol analysis for protocol aware network functions can be implemented. A good future work would be to use the DPI to extend OpenFlow based switches - use the tags created by the DPI service to drive policies in conventional switches.

Appendix

Code Repository

The code repository for the project is deployed at: https://github.com/dhruv-jain/clickos_ids

Acknowledgment

The authors would like to thank Dr. Hongxin Hu, Hongda Li and Nishant Vishwamitra for their help.

References

[1] https://en.wikipedia.org/wiki/Xen

[2] ClickOS and the Art of Network Function Virtualization: Joao Martins, Mohamed Ahmed, Costin Raiciu and Vladimir Olteanu

[3] Enabling Fast, Dynamic Network Processing with ClickOS by: Joao Martins, Mohamed Ahmed, Costin Raiciu and Felipe Huici

[4] https://bitbucket.org/hongdal/clickos-setup/wiki/Tutorial%20of%20installing%20ClickOS%20testbed

[5] Programmable Networking with Open vSwitch: Jesse Gross

[6] https://en.wikipedia.org/wiki/Deep_packet_inspection

[7] http://www.secdev.org/projects/scapy/doc/introduction.html#about-scapy

[8] https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/what-is-floodlight-controller/

[9] http://jakeboxer.com/blog/2009/12/13/the-knuth-morris-pratt-algorithm-in-my-own-words/