IK2220 - SDN & NFV - Phase 2

The target of the course's assignment is to involve students in modern networking design and implementation using Software Defined Networking (SDN) and Network Functions Virtualization (NFV) principles.

Description

The goal of the assignment is to give you hands-on experience in practical SDN & NFV implementations. You should learn how to:

- Emulate network infrastructure,
- Generate traffic patterns using well-known tools,
- Launch and program an SDN controller,
- Instruct the data plane devices using both SDN and NFV techniques,
- Capture the state of any device in the network,
- Capture traffic to inspect the message exchanges,
- Implement advanced network functions (i.e., firewall, load balancer, NAPT, and IDS).

This assignment will be split into two phases.

- Phase I (Mininet+SDN+Testing)
- Phase II (NFV+Testing)

Phase 2 (SDN+NFV)

In the first phase, we used SDN tools to implement a simplified version of a cloud topology depicted in Figure 1. In the second phase, we will complete the topology by adding four new nodes, as follows:

- A load balancer (**lb1**) is added between the core switch (sw2) and the three web servers ensuring that incoming requests, which target the virtual IP, will be modified accordingly with the destination IP address of a server in a round-robin fashion. Switch sw4 is used to connect the load balancer with the Web cluster.
- An Intrusion Detection System (**IDS**) module is added before lb1 to inspect the incoming packets. This module will search at the incoming packets' payload for certain "suspicious" patterns. If such a pattern is identified, the packet will be redirected to the inspector (**insp**) server for further processing. Otherwise, legal traffic will pass through.
- A Network Address and Port Translator (**NAPT**) that translates the private IP addresses of hosts in PrZ into public IP addresses within the range of DmZ and PbZ.

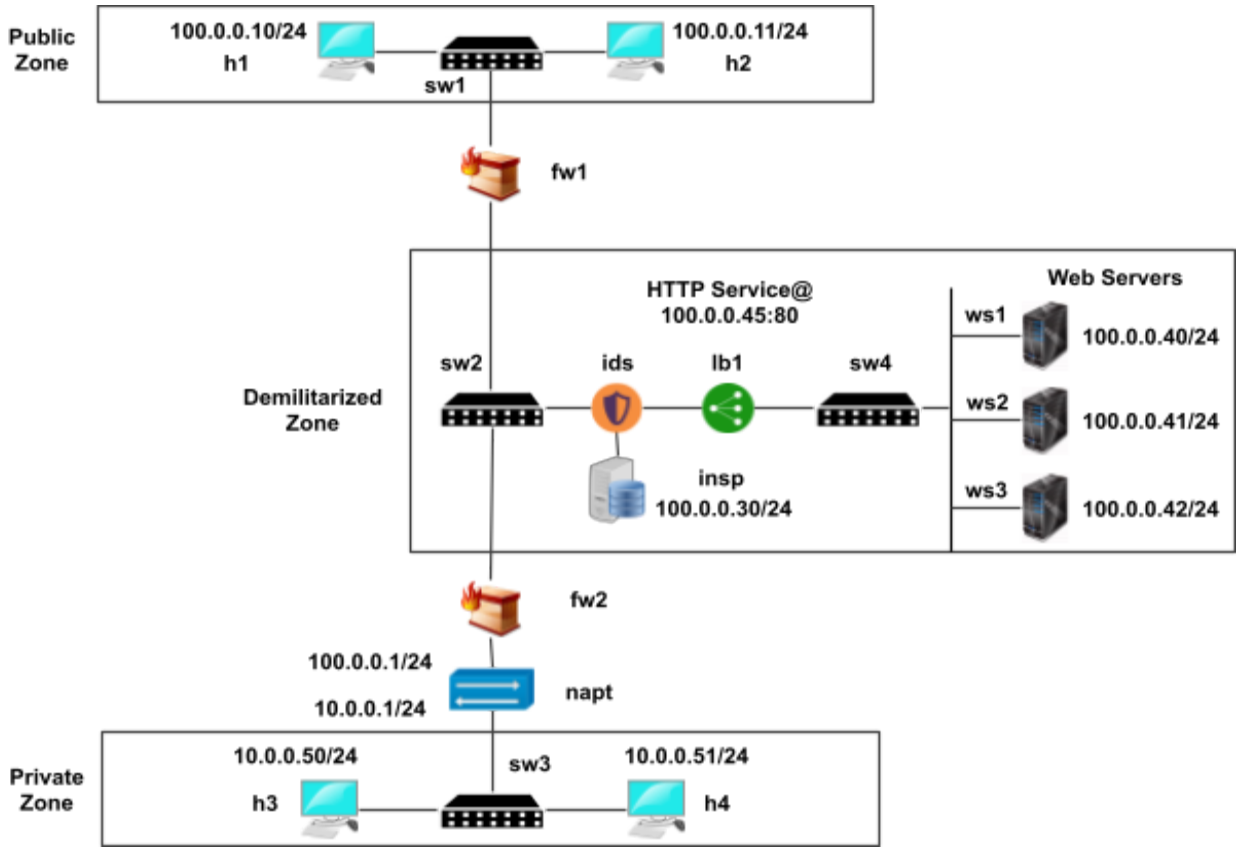

Figure 1: Cloud Topology with three main zones. A public zone(PbZ) that sits close to the Internet, a protected, demilitarized zone (DmZ) that contains servers (Web) and a private zone (PrZ) that contains cloud resources (e.g. VMs).

Topology

You should first add the new nodes to the topology. The NFV elements of the topology (i.e., lb1, ids, and napt) can be added as SDN switches with a simple MAC learning functionality and L2 forwarding abilities. Later, you will use the Click NFV framework to realize more advanced functions on these elements. This can happen by instructing Click to read/write packets from/to the interfaces of a target Mininet switch (e.g., lb1) using FromDevice and ToDevice Click elements, respectively. In between these elements, you should then construct the Click pipeline that is required to realize a given network function (e.g., load balancing). Note that in this phase, the nodes of PrZ must change IP configuration since the presence of NAPT will provide IP address translation from one subnet (10.0.0/24 of PrZ) to another (100.0.0/24 of DmZ/PbZ).

The topology of phase 2 should comply with the following rules:

1. The private zone sits behind a NAPT, hence the IP addresses of h3 and h4 are not visible to the outside world (DmZ, PbZ). The NAPT must apply the Source NAPT function to the outbound traffic (from PrZ) and the Destination NAPT function to the inbound traffic (towards PrZ).
2. DmZ should provide virtual services to both PrZ and PbZ. Specifically:
    a. HTTP/TCP service will be accessible on IP address 100.0.0.45/24 and port 80. This IP does not exist in Figure 1 because it is a virtual Service IP. Load balancer lb1 is responsible to handle this virtual IP. When lb1 receives a packet with destination 100.0.0.45 it must redirect the packet to one of the three web servers. This means that destination IP address 100.0.0.45 should be translated into the IP address of the selected server and then forwarded. In the opposite direction, when lb1 receives a packet from one of the three servers, it must change the source IP address to 100.0.0.45 (virtual Service's IP) such that the host does not understand the existence of the servers behind this service. This functionality also implies the existence of NAPT, besides load balancing. In addition, the existence of IDS before lb1 puts some restrictions on the content of the HTTP requests. Specifically, the IDS module must access the payload of incoming requests and search for patterns that imply (a) code injection and (b) dangerous use of HTTP methods. If such a pattern is found, the packet will be redirected to the inspector (server insp) for further inspection. More details are provided below.
    b. Users from PbZ and PrZ must be able to (successfully) ping the virtual IPs (i.e., the load balancers must generate ICMP responses).
    c. The rest of the traffic from PbZ and PrZ to DmZ must be blocked.
3. PbZ nodes should be accessible from anywhere.
4. The inspector server is a passive node of this topology and does not require any services or communication with other nodes. The link between IDS and the inspector must simply be always on in order for the IDS to push the "suspicious" packets there.

Phase 2 - Implementation Details

To realize the above requirements, you should instruct all the data-plane nodes to forward/drop traffic accordingly. The instruction will be done using **both** SDN and NFV techniques. The following list describes each network element's functionality separately:

1. Switches sw1-sw4 are regular OpenFlow v1.0 [3] L2 learning switches. Use the L2 learning module of POX to instruct the appropriate rules. (Phase 1)
2. Firewalls fw1 and fw2 are OpenFlow v1.0 L2 learning switches extended with stateful access control rules (sent by the SDN controller). These rules realize the functionality of DmZ and PrZ described above. Use the L2 learning module of POX as a basis to extend each firewall. (Phase 1)
3. The load balancer, ids, and napt must be implemented in Click [6]. This means that you should instruct Click to capture packets from the interfaces of the Mininet switches lb1, ids, and napt, respectively. POX should not send any OpenFlow rules to these switches because their functionality will be programmed using the Click language. POX will only get registration events from these Mininet nodes (when Mininet boots). Once POX gets such an event, it should simply start the Click module responsible for this node.

To start Click modules in POX, you can simply use subprocess.Popen() to run the proper click module.

4. In the following paragraphs, we explain the Click functions:
    a. Load Balancer (lb1): First, the load balancer must read packets from the interface and classify the packets into four basic classes. ARP requests, APR replies, IP packets, other packets. Upon an ARP request (that targets the virtual IP of the corresponding service), an ARP reply must be generated using an ARPResponder element per interface. This reply must contain the MAC address of the virtual service. Of course, this MAC address will be virtual as well, but the hosts should be able to generate IP packets after getting back an APR reply from the load balancer. ARP responses must be sent to ARPQuerier elements (one per interface). IP packets must be sent to a pipeline of elements that will realize the load balancing procedure explained above. There are two directions for the load balancer, one towards the servers and another towards the clients. Hence two pipelines are required as we explain below. Finally, packets that are neither ARP nor IP must be discarded.

    IP pipeline towards the servers: A classifier/filter must be applied to packets coming from PbZ and have destination IPs different from the virtual IP. After this classifier, a modification element (i.e., IPRewriter) should work in tandem with RoundRobinIPMapper in order to write the correct destination addresses (of one server) to the incoming packet.

IP pipeline towards the clients: The IPRewriter element above should also translate the source address of the servers into the virtual addresses that the load balancer possesses. That way, the client cannot notice the existence of this “proxy” node.

5. IDS (ids) captures packets from the interfaces of the Mininet switch ids. No IP address is assigned to this module hence it should be acting as a forwarder of the traffic. Specifically, ARP frames, ICMP ping requests, and responses, as well as TCP-signaling must traverse the IDS transparently.

   There are two kinds of analysis you must perform, both on HTTP packets. You must classify IP packets using IPClassifier to only match HTTP traffic.

   a) The first pattern we want to inspect is the HTTP method of each request.
      You have to check which HTTP method is used by the user. A Classifier element must be used to classify all the possible traffic patterns that might pass through the IDS and search for patterns in the HTTP packets. Specifically, HTTP provides the following methods.
      - GET and POST are the most common methods used to request a web page. GET passes any parameters via the URL while POST parameters are sent in the HTTP payload. This makes POST safer.
      - HEAD method is similar to GET, but the server returns only the headers as a response.
      - OPTIONS method asks the server which methods are supported in the web server. This provides a means for an attacker to determine which methods can be used for attacks.
      - TRACE method allows the client to see how its request looks when it finally makes it to the server. An attacker can use this information to see if any changes are made to the request by firewalls, proxies, gateways, or other applications.
      - PUT method is used to upload resources to the server. This method can be exploited to upload malicious content.
      - DELETE method is used to remove resources from the server. Similarly, it can be exploited to delete useful content.
      - CONNECT method can be used to create an HTTP tunnel for requests. If the attacker knows the resource, he can use this method to connect through a proxy and gain access to unrestricted resources.

      Your IDS module must only allow POST and PUT methods. This means that you should identify the exact location of the HTTP method field in the header space and use the Classifier to detect the hexadecimal values of the method asked by the user. Only if the method is POST or PUT the packet is forwarded to lb1; otherwise, the packet is sent to the inspector.

   b) The second interesting pattern is Linux and SQL code injection via the HTTP PUT method. The very first bytes of the payload after the HTTP header must be matched against the the following keywords:
      i. cat /etc/passwd
      ii. cat /var/log/

iii. INSERT
iv. UPDATE
v. DELETE

The Search element will be useful to advance the "packet payload" pointer to the first byte of the HTTP payload. Check the HTTP header format to find a way to pass the header and "jump" directly to the payload. Do not use Search to look for the keyword themselves, it would be very inefficient because you only want to look at the very first byte of the payload while search will check for the pattern starting at any bytes of the packet. Use Classifier and the hexadecimal values of those patterns instead.

6. The NAPT will also be implemented as a Click module that captures packets from the interfaces of the Mininet switch napt. This module must handle ARP properly and apply address and port translation on TCP and ICMP packets. For TCP, you should use the IPRewriter element, while for ICMP an ICMPPingRewriter element can be used to translate only ICMP echo requests and responses (you can safely omit other ICMP packets). Both traffic directions must end up at these translators which will convert 10.0.0/24 addresses into 100.0.0/24 and vice versa. The IP address of the DmZ interface of the NAPT is 100.0.0.1 and the IP address of its PrZ interface is 10.0.0.1 as shown in Figure 1.

Finally, the inspector server will be a normal Mininet host that captures packets from its interface (e.g., using tcpdump) and dumps the packets to a PCAP file. This file will be used as proof to assess the correct behavior of the IDS module. The PCAP file does not need to be sent to us, but the system to capture packets must be put in place.

Phase 2 - Testing

In this phase, you should extend your tests from phase 1 with more tests. For example, in phase 2, we have the concept of virtual IPs so one of your new tests must ping these IPs and parse the responses.

Besides the automatic tests that will stress your entire application, you should also deliver a set of files that assess the functionality of each of your Click-based network functions. To do so, you should use the AverageCounter and Counter elements in order to measure:

1. The number of packets read and written by each Click module as well as the observed packet rate (throughput). These counters must be placed right after each FromDevice and right before each ToDevice Click element. Use the AverageCounter element.
2. If your Click module classifies traffic, you must count the number of packets observed per traffic class. See the example of lb1 in Table 1. The number of dropped packets must be included since it is also a traffic class. Use the Counter element in this case.

After your automated tests above have stressed the entire application, you should tear down your POX module as well as all your Click modules. Once a Click module terminates, it can use the DriverManager element to print out the collected counters to a designated file. Each Click network function should generate a file named *<function ID>.report*. For example, lb1.click should generate a file lb1.report.

```
=================== LB1 Report ===================
   Input Packet rate (pps): 0.262347351823
  Output Packet rate (pps): 0.89408388857

Total # of   input packets: 395
Total # of  output packets: 258

Total # of   ARP  requests: 13
Total # of   ARP responses: 4

Total # of service packets: 145
Total # of    ICMP packets: 4
Total # of dropped packets: 51
=================================================
```

Table 1: Example format of counters reported by load balancer 1 (lb1.report).

Phase 2 - Deliverable

1) Use the same directory structure as you had in phase 1. In the application folder, add a subfolder `nfv` where you need to place the Click implementation of the load balancer, the IDS, and the NAPT.

2) Your SDN application should now call the Click scripts above, hence you need some minimal modifications to your SDN controller.

3) When you are ready to submit strictly follow the structure below:

- Create a folder with name ik2220-assign-phase2-team<Number>
    - E.g., ik2220-assign-phase2-team1
- Create a file MEMBERS stating the name and email of each member of the team.
- Create a Makefile with the same semantics than phase 1
- Use the very same ‘topology’ subdirectory as in phase 1
- Use the very same ‘application’ subdirectory as in phase 1
    - Create a subdirectory ‘nfv’
    - Put your NFV sources into directory ‘nfv’
- In the subdirectory ‘results’
    - include the tests that stress your entire application.
    - include 3 .report files, each one corresponding to a Click-based network function (see above).
        - E.g., lb1.report
- Make a tarball of the folder ik2220-assign-phase2-team<Number>(.tar.gz)
- Upload it before:
    - **May 23, 16:59**.

4) Your deliverable in this phase must contain the full functionality of the assignment (both SDN and NFV parts).

Note that all of the assignments will be checked with a plagiarism tool.
You will face a large penalty if we detect any case!

Grading

Given that you have successfully submitted your assignment in time (before the deadline) and you have followed the restrictions that we give you above, then:

1. NFV applications' design and implementation (45/100)
    a. If your NFV applications are correct you get (30/100)
        i. Load balancer gets (10/100)
        ii. NAPT gets (10/100)
        iii. IDS gets (10/100)
    b. If your design is correct you get the rest (15/100)
2. Tests get (15/100).

Phase 2 will account for 60% of the final assignment mark.

Useful Resources for Learning Click

You can learn more about Click from the following resources:

1. https://github.com/tbarbette/fastclick/wiki/Tutorial
2. https://www.bo-yang.net/2015/01/07/click-notes-click-language

References

1. Mininet network emulator: http://mininet.org/
2. POX Wiki: http://intronetworks.cs.luc.edu/auxiliary_files/mininet/poxwiki.pdf
3. OpenFlow: https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf
4. OpenVSwitch: https://github.com/openvswitch/ovs
5. Wireshark: https://www.wireshark.org/
6. POX Manual https://noxrepo.github.io/pox-doc/html/

Notes on Click

By default, the FromDevice element will take a copy of the packet, meaning that the original packet will continue in the Linux networking stack. This means that you could see duplicate

packets, that will lead to a complete mess. You must use the SNIFFER false parameter to destroy the original packet. See https://github.com/kohler/click/wiki/FromDevice.u for more details.

You can see the documentation for all elements by using the command `man Element`. For instance, *`man Search`* will give the documentation for the *Search* element.

Enable Internet access on the VM

Remember your tests should work with a clean VM. Do not add any dependencies and do not rely on anything you install. You're not allowed to use non-presented libraries anyway.

You may want to update tmux, or install your preferred text-based editor on your VM. To do so, you may enable the access like this.

On your VM do:

```
sudo ip route add default via 192.168.10.2
```