# Testing AWS Lab Setup via Packet Sniffing

## 1 Overview

Packet sniffing is an important concept in network security; it is a major threat in network communication. Being able to understand these this threat is essential for understanding security measures in networking. There are many packet sniffing tools, such as `Wireshark`, `Tcpdump`, `Netwox`, `Scapy`, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing is implemented in software. There are three objectives for this lab

1. get used to the AWS environment
2. learn to use the sniffing tool
3. write simple sniffing and snooping programs

## 2 Environment Setup

In this lab, you will use two machines that are connected to the same LAN. Figure 1 depicts the lab environment. You will monitor from the attacker machine while you ping from the user machine.

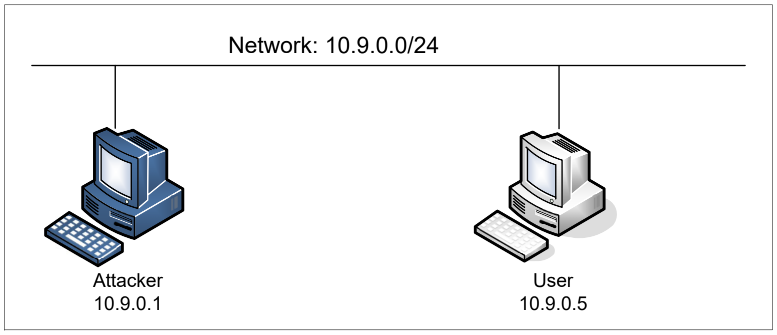


Figure 1: Lab environment setup

**NOTE: The network setup is done via scripts. Please carefully follow the directions below.**

### 2.1 Network Setup

For this lab you will need two terminal windows in your Kali box open. To setup the environment, complete the following steps:

**Creating the Network in Terminal 1**

1. Open a terminal window
2. SSH into your assigned AWS virtual machine via *ssh -l ubuntu* assigned ip
3. Execute **./startLab.sh** - this will (1) change the directory to the Lab0 directory, (2) change to the superuser *seed*, (3) pull the docker images and create the network, and (4) leave you as the *seed* user in the terminal window.

**Writing code from Terminal 2**

1. Open a second terminal window
2. SSH into your assigned AWS virtual machine via *ssh -l ubuntu* assigned ip
3. Change directory to *∼/Documents/Labs/Lab0*

## 2.2 About the Attacker Container

In this lab, you will use the attacker container as the monitoring machine. This attacker container, executed from the *seed* account, is configured per the following:

- *Shared folder.* To launch attacks your code will need to be placed inside the attacker container. You will write our code in the `./volumes` folder (from Terminal 2), so it can be used inside the containers. NOTE: this code is written using the *ubuntu* user.

```
volumes:
        - ./volumes:/volumes
```

- *Host mode.* In this lab, the attacker needs to be able to sniff packets, but running sniffer programs inside a container has problems, because a container is effectively attached to a virtual switch, so it can only see its own traffic, and it is never going to see the packets among other containers. To solve this problem, we use the `host` mode for the attacker container. This allows the attacker container to see all the traffics. The following entry used on the attacker container:

```
network_mode: host
```

  When a container is in the `host` mode, it sees all the host's network interfaces, and it even has the same IP addresses as the host. Basically, it is put in the same network namespace as the host VM. However, the container is still a separate machine, because its other namespaces are still different from the host.

### Getting the network interface name.

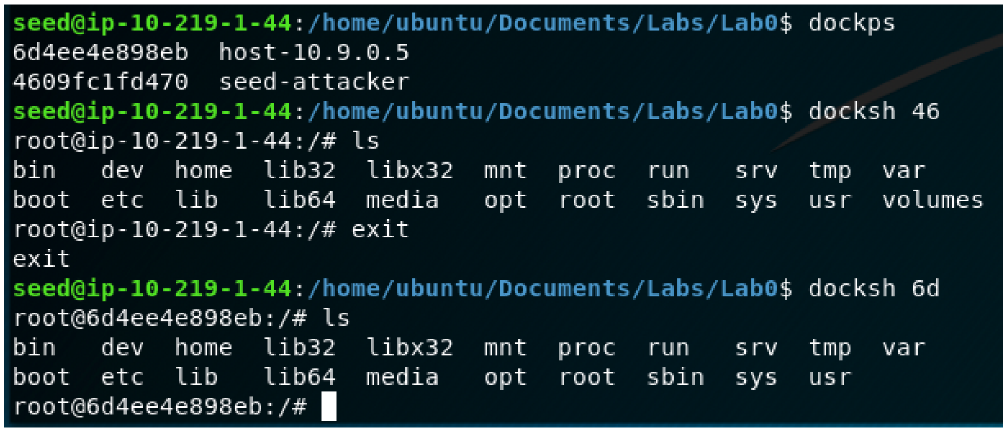
When we use the provided Compose file to create containers for this lab, a new network is created to connect the VM and the containers. The IP prefix for this network is `10.9.0.0/24`, which is specified in the `docker-compose.yml` file. The IP address assigned to our VM is `10.9.0.1`. We need to find the name of the corresponding network interface on our VM, because we need to use it in our programs. The interface name is the concatenation of `br-` and the ID of the network created by Docker. When we use `ip addr` to list network interfaces, we will see quite a few. Look for the IP address `10.9.0.1`.

```
$ ip addr
br-c93733e9f913: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        ...
```

## 3 Basic Commands

Now that you have the environment setup let's execute a few commands. Recall that all docker command are executed from the *seed* account.

- *dockps* - lists the ID of the container and the container name
- *docksh* first two of the container ID - opens a sh terminal as root for that container



```
seed@ip-10-219-1-44:/home/ubuntu/Documents/Labs/Lab0$ dockps
6d4ee4e898eb  host-10.9.0.5
4609fc1fd470  seed-attacker
seed@ip-10-219-1-44:/home/ubuntu/Documents/Labs/Lab0$ docksh 46
root@ip-10-219-1-44:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@ip-10-219-1-44:/# exit
exit
seed@ip-10-219-1-44:/home/ubuntu/Documents/Labs/Lab0$ docksh 6d
root@6d4ee4e898eb:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@6d4ee4e898eb:/#
```

Figure 2: Examining the Docker containers

Notice, in Figure 2, the host machine doesn't have a volumes directory while the *seed-attacker* does have a volumes directory. This is because the volumes directory in *seed-attacker* is linked to the volumes directory in Lab0 for the *ubuntu* user.

## 4 Lab Task Set 1: Using Scapy to Sniff Packets

Many tools can be used to do sniffing, but most of them only provide fixed functionalities. Scapy is different: it can be used not only as a tool, but also as a building block to construct other sniffing tools, i.e., we can integrate the Scapy functionalities into our own program. In this task, we will use Scapy for each task.

To use Scapy, we can write a Python program, and then execute this program using Python. See the following example. We should run Python using the root privilege. At the beginning of the program (Line ①), we should import all Scapy's modules.

```
# view mycode.py
#!/usr/bin/env python3

from scapy.all import *     ①

a = IP()
a.show()

# python3 mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  ...


// Make mycode.py executable (another way to run python programs)
# chmod a+x mycode.py
# mycode.py
```

We can also get into the interactive mode of Python and then run our program one line at a time at the Python prompt. This is more convenient if we need to change our code frequently in an experiment.

```
# python3
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version   = 4
  ihl       = None
  ...
```

## 4.1 Task 1.1: Sniffing Packets

Wireshark is the most popular sniffing tool, and it is easy to use. We will use Wireshark in a different lab. The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs. Sample code is provided in the following:

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
  pkt.show()

pkt = sniff(iface='br-c93733e9f913', filter='icmp', prn=print_pkt)
```

The code above will sniff the packets on the `br-c93733e9f913` interface. You will need to change this interface for your specific machine. In the terminal window of the *seed* user, issue the command **ip addr**. This command will display the information about your network devices. Find the network device that contains *br-some hex value*.

To write code for this section of the lab:

1. In the *ubuntu* user window change directory to volumes
2. Using nano open the file sniffer.py
3. Type the code from the listing above into sniffer.py
4. Change the bridge device *iface='br-c93733e9f913'* to the value of your bridge device which you found from *ip addr*. **NOTE: You will need to type this and not copy and paste because the single quotes won't be correct otherwise.**
5. Press ctrl-x Y Enter to save the file
6. Execute chmod 777 ./sniffer.py

**To test your code follow the steps below. NOTE: you will need both terminals.**

In the *seed* account terminal

1. Login into the *seed-attacker* container via *docksh* and the first two values from the *seed-attacker* container id from the *dockps* command.
2. Change directory to volumes
3. Execute ./sniffer.py

In the *ubuntu* account terminal

1. Execute the command **sudo -su *seed***
2. Login into the user machine via *docksh* and the first two values from the user container id from the *dockps* command.
3. Execute the command **ping -c 5 10.9.0.1**
4. Watch what happens in the *seed-attacker* terminal window.

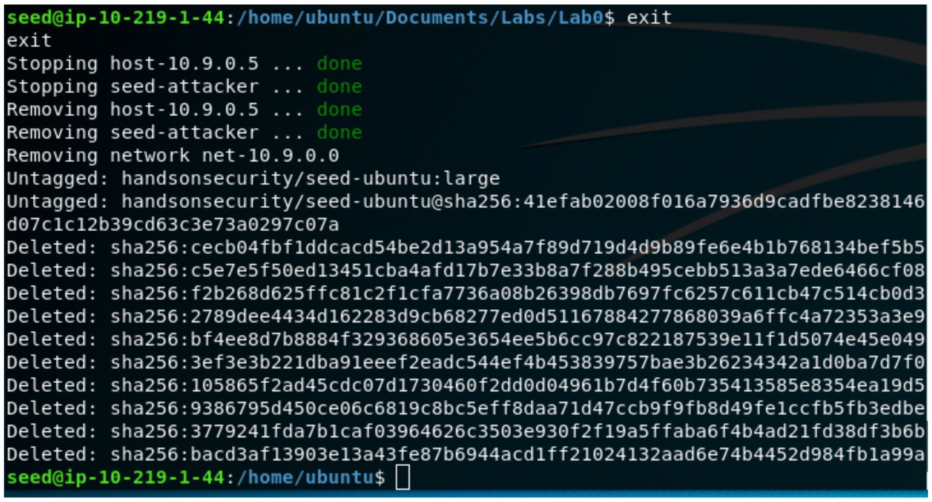
**Grab screen captures of both the ping command from the *user* machine and the executing sniffer.py from the *seed-attacker* machine.**

## 5 Clean Up

From the *user* machine where you executed the ping command, execute exit until you log out of the AWS VM, and then close the terminal window.

From the *seed-attacker* machine where you executed sniffer.py

1. Press ctrl-c
2. Execute exit (possibly multiple times) until you get back to the seed@ip-10.219.1.# command line.
3. Execute exit - This exit should produce an image similar to Figure 3.



```
seed@ip-10-219-1-44:/home/ubuntu/Documents/Labs/Lab0$ exit
exit
Stopping host-10.9.0.5 ... done
Stopping seed-attacker ... done
Removing host-10.9.0.5 ... done
Removing seed-attacker ... done
Removing network net-10.9.0.0
Untagged: handsonsecurity/seed-ubuntu:large
Untagged: handsonsecurity/seed-ubuntu@sha256:41efab02008f016a7936d9cadfbe8238146
d07c1c12b39cd63c3e73a0297c07a
Deleted: sha256:cecb04fbf1ddcacd54be2d13a954a7f89d719d4d9b89fe6e4b1b768134bef5b5
Deleted: sha256:c5e7e5f50ed13451cba4afd17b7e33b8a7f288b495cebb513a3a7ede6466cf08
Deleted: sha256:f2b268d625ffc81c2f1cfa7736a08b26398db7697fc6257c611cb47c514cb0d3
Deleted: sha256:2789dee4434d162283d9cb68277ed0d51167884277868039a6ffc4a72353a3e9
Deleted: sha256:bf4ee8d7b8884f329368605e3654ee5b6cc97c822187539e11f1d5074e45e049
Deleted: sha256:3ef3e3b221dba91eeef2eadc544ef4b453839757bae3b26234342a1d0ba7d7f0
Deleted: sha256:105865f2ad45cdc07d1730460f2dd0d04961b7d4f60b735413585e8354ea19d5
Deleted: sha256:9386795d450ce06c6819c8bc5eff8daa71d47ccb9f9fb8d49fe1ccfb5fb3edbe
Deleted: sha256:3779241fda7b1caf03964626c3503e930f2f19a5ffaba6f4b4ad21fd38df3b6b
Deleted: sha256:bacd3af13903e13a43fe87b6944acd1ff21024132aad6e74b4452d984fb1a99a
seed@ip-10-219-1-44:/home/ubuntu$ 
```

Figure 3: Cleaning up the containers

## 6 Submission

Submit a single PDF containing

- Screen capture of the *dockps* command
- Screen capture of the ping command from the *user* machine.
- Screen capture of the running sniffer.py from the *seed-attacker* machine.
- Screen capture of the *seed-attacker* machine after clean up.

Name your PDF your last name first letter of your first name Lab0.pdf.
(Example: steinersLab0.pdf).