Laboratory Exercise: Full Packet Capture

Goals

1. How to do a full packet capture and how to do field-level filtering while capturing.
2. How to build a filtering sensor using nc and sed/grep/awk.
3. How to build a persistence service to store packets.
4. How to build a basic GUI interface to visualize captured packet data (Optional)

Prerequisites

Before you can begin with this particularly laboratory, you will need to set up a virtual network environment with several different virtual machines with which a number of different software packages will be installed.

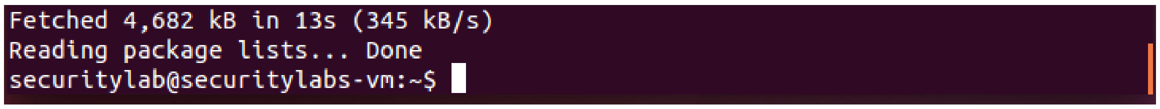
Setting up the Ubuntu VM's

Using the Ubuntu VM that you should already have from previous labs, go ahead and update all the software packages on that VM. To do this update step, your Ubuntu Vms will need to have their Network Interface Cards configured to be attached to the Bridged interface (so they can use the host's network adapter to go out onto the Internet).

You can do this by running the following command:

```
$ sudo apt-get update
```

Once it's successfully updated you should receive the following message:



```
Fetched 4,682 kB in 13s (345 kB/s)
Reading package lists... Done
securitylab@securitylabs-vm:~$
```

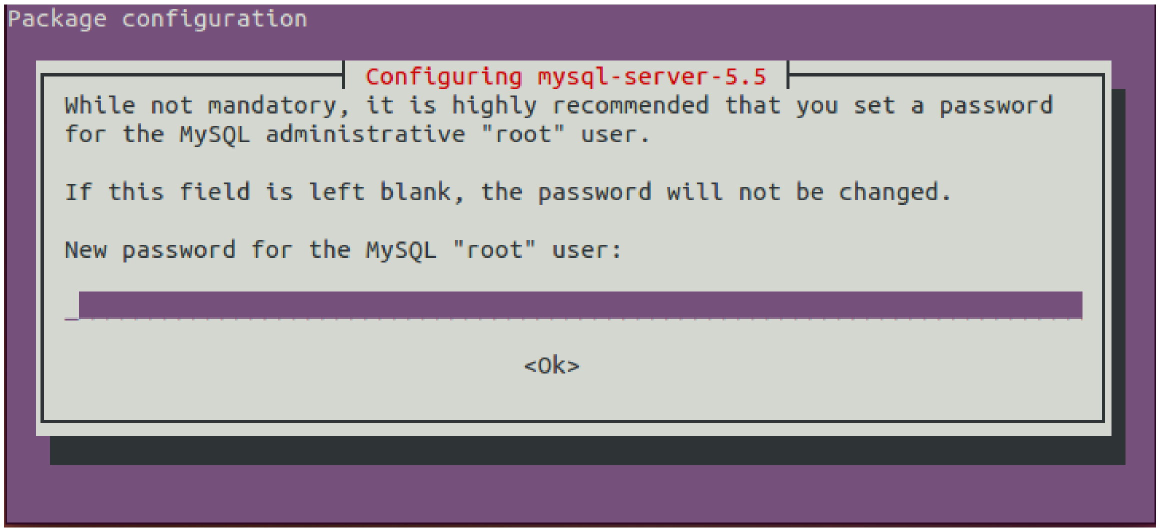
Now you can begin installing the software that is required. The software packages that you will need to install are: tsch, mysql-server and tshark. To do this, enter the following commands in the terminal:

```
$ sudo apt-get install tcsh
```

This should just install automatically without any issues.

```
$ sudo apt-get install mysql-server
```

During the installation you will be informed that it will require x amount of additional disk space and whether you should accept, type Y when prompted. Once that happens it will begin the installation and it will then prompt you to create a root password, as shown on the following page.



```
Package configuration

Configuring mysql-server-5.5

While not mandatory, it is highly recommended that you set a password
for the MySQL administrative "root" user.

If this field is left blank, the password will not be changed.

New password for the MySQL "root" user:

_

<Ok>
```

Type in a password. retype it and then continue with the installation.

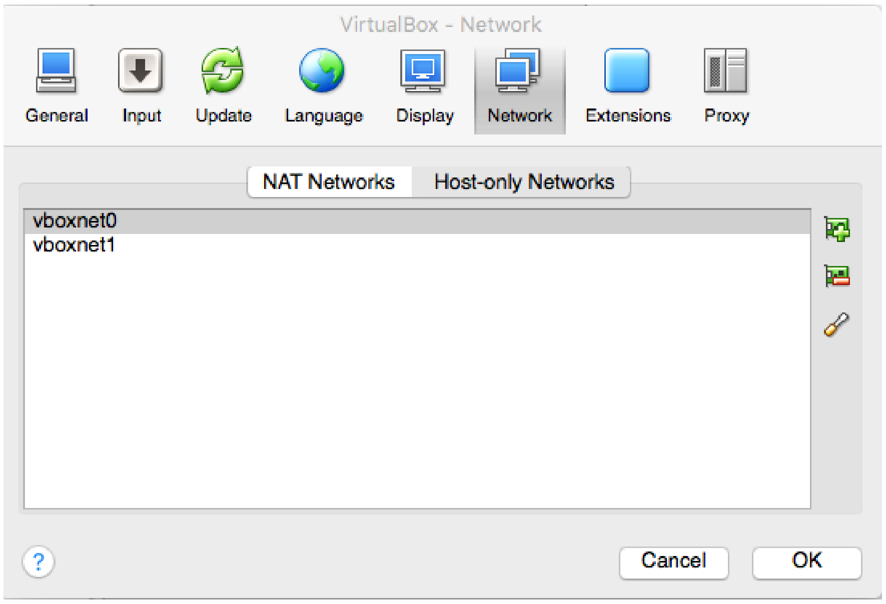
```
$ sudo apt-get install tshark
```

This should also just install without any prompts.

The Ubuntu VM has now been updated with all the prerequisites. You should shut down that VM. In VirtualBox, clone the entire VM and all its dependant files. Rename that second instance that you have just created to Ubuntu-LAN so you can distinguish between the two Ubuntu VM's.

With the setup of the two Ubuntu machines which will act as your 1) internal machine on your LAN and 2) a machine accessible via the WAN, now you need to create your firewall VM.

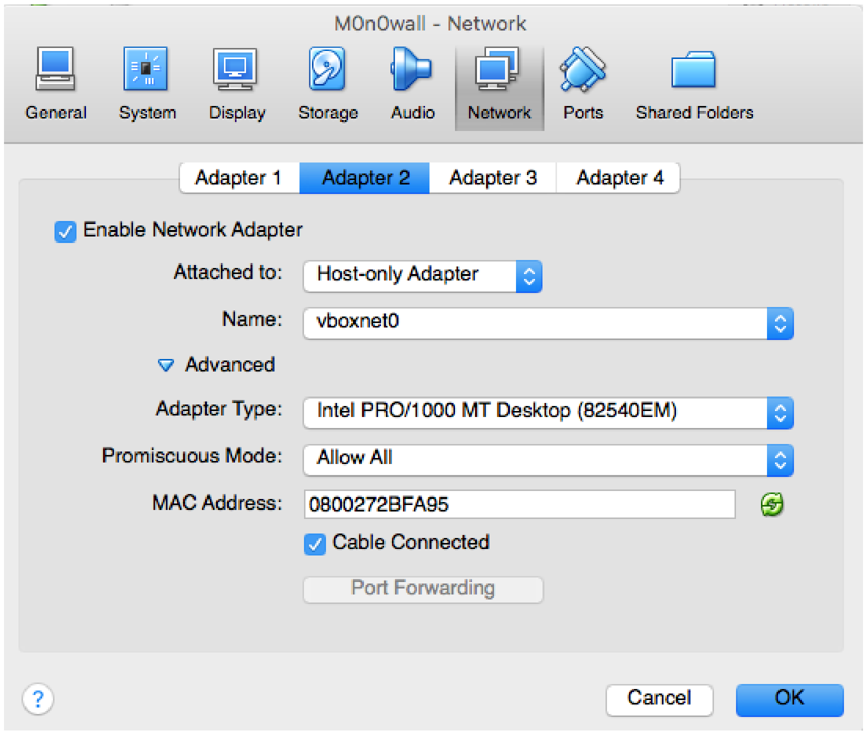
Before you can go ahead and create your M0n0wall machine you need to first create two Host-Only Network interfaces. According to the VirtualBox manual[1] you need to go to "File" -> "Preferences" -> "Network" -> "Host-only network" -> "(+)Add host-only network". Once you have created it, go ahead and create a second and then you should see something like that shown on the following page.



vboxnet0
vboxnet1

Go ahead and edit your M0n0wall VM with two network interface cards (NIC) – one for the LAN and the other for the WAN. It is important that you also put the network adapter into Promiscuous mode, as shown below. Now load up the M0n0wall VM.

1 https://www.virtualbox.org/manual/ch06.html#network_hostonly



Enable Network Adapter ✓

Attached to: Host-only Adapter

Name: vboxnet0

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

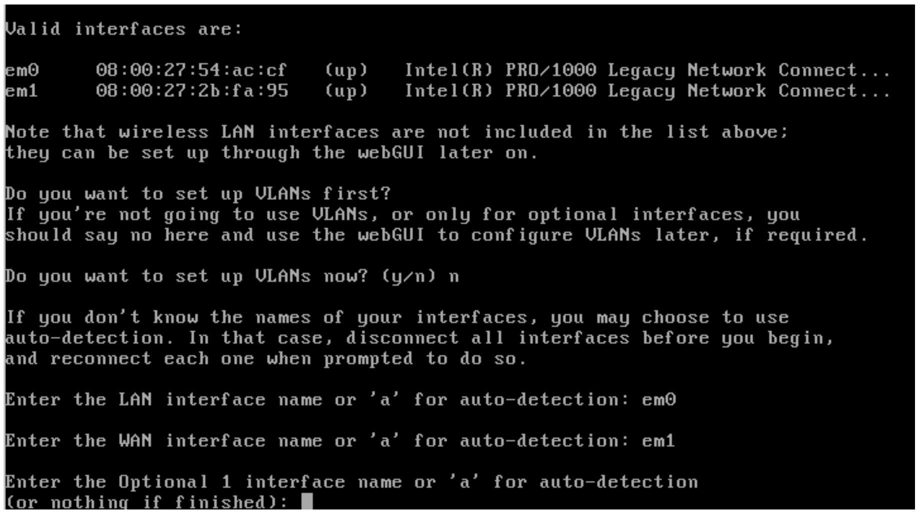
Promiscuous Mode: Allow All

MAC Address: 0800272BFA95

Cable Connected ✓

M0n0wall Interfaces

The next step is to ensure that both M0n0wall interfaces are setup with the correct valid interfaces. You do this by selecting option 1 once it's booted up. It will ask you if you want to setup a VLAN so type 'n' for this option. It will then ask you to set up the LAN interface so select the first option, and select the second option for the WAN. The final stage is to say yes and restart the M0n0wall so it can save the changes, as shown below:



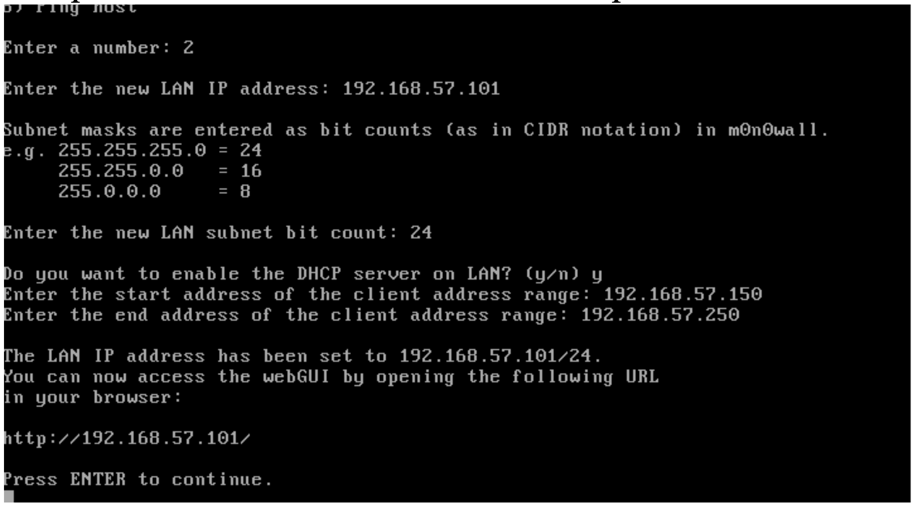
```
Valid interfaces are:

em0     08:00:27:54:ac:cf   (up)   Intel(R) PRO/1000 Legacy Network Connect...
em1     08:00:27:2b:fa:95   (up)   Intel(R) PRO/1000 Legacy Network Connect...

Note that wireless LAN interfaces are not included in the list above;
they can be set up through the webGUI later on.

Do you want to set up VLANs first?
If you're not going to use VLANs, or only for optional interfaces, you
should say no here and use the webGUI to configure VLANs later, if required.

Do you want to set up VLANs now? (y/n) n

If you don't know the names of your interfaces, you may choose to use
auto-detection. In that case, disconnect all interfaces before you begin,
and reconnect each one when prompted to do so.

Enter the LAN interface name or 'a' for auto-detection: em0

Enter the WAN interface name or 'a' for auto-detection: em1

Enter the Optional 1 interface name or 'a' for auto-detection
(or nothing if finished):
```

Setting up the LAN

The LAN interface should have the IP address 192.168.57.101 that you will configure during the initial setup of the machine. Do this by selecting option 2 and assigning the new IP address.

DHCP Server setup

Part of the task is to setup a DHCP server on the firewall LAN side so that the Ubuntu LAN machine is assigned an IP address via DHCP from the M0n0wall machine. You should setup the DHCP server on the M0n0wall and assign it a range of 192.168.57.150 - 192.168.57.250. The setup for both the LAN and DHCP setup is shown below:



```
Enter a number: 2

Enter the new LAN IP address: 192.168.57.101

Subnet masks are entered as bit counts (as in CIDR notation) in m0n0wall.
e.g. 255.255.255.0 = 24
     255.255.0.0   = 16
     255.0.0.0     = 8

Enter the new LAN subnet bit count: 24

Do you want to enable the DHCP server on LAN? (y/n) y
Enter the start address of the client address range: 192.168.57.150
Enter the end address of the client address range: 192.168.57.250

The LAN IP address has been set to 192.168.57.101/24.
You can now access the webGUI by opening the following URL
in your browser:

http://192.168.57.101/

Press ENTER to continue.
```

Due to rules that have to be setup it is necessary for the Ubuntu LAN machine to always have the same IP address. You should therefore add a DHCP reservation on the M0n0wall with the Ubuntu LAN's MAC Address and assigned it the IP address 192.168.57.254. To do this, load up the Ubuntu-LAN machine (make sure that the network interface of this machine is set to vboxnet0). On the terminal issue the `ifconfig` command and make a note of the MAC address (`HWaddr`). Now open up a browser and type in the IP address of the M0n0wall LAN interface. It will load up the web GUI of the M0n0wall machine and prompt you to login with your credentials. Use the following:

```
username: admin
password: mono
```

Now browse to Services > DHCP Server > Reservations > Add Mapping. Fill out the required information that includes the MAC address that you just made a note of, the IP address 192.168.57.254 and a description of what it is.

Save and apply the new settings and reboot the Ubuntu LAN machine. It should come back up with the correct settings, including the Ubuntu-LAN default gateway, which should be set to 192.168.57.101 (the IP address of the M0n0wall LAN side). Check this to ensure that it is working correctly.

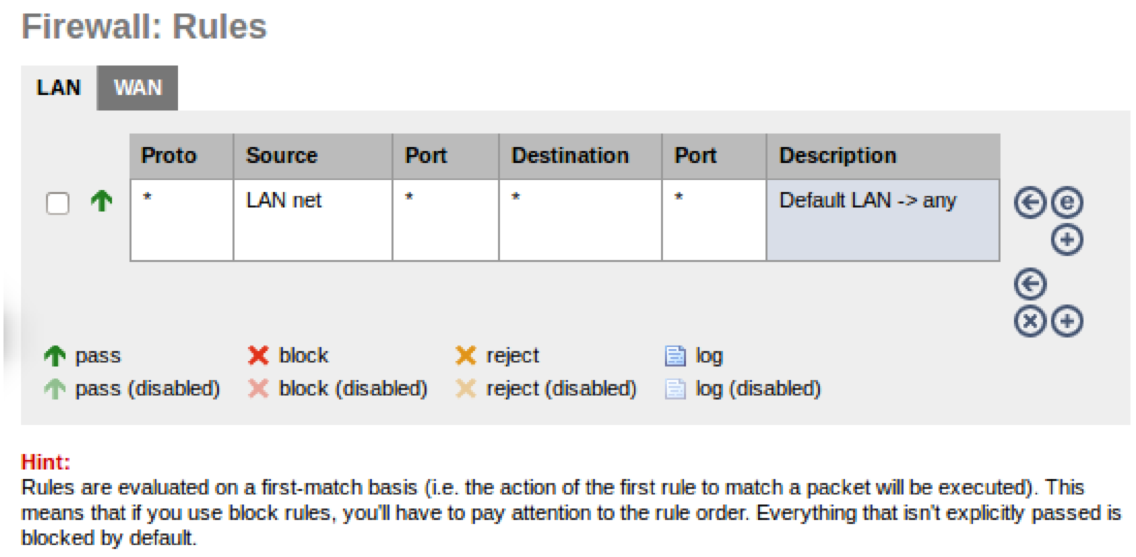
WAN setup

Setting up the WAN interface is slightly more challenging. You will have to configure it through the M0n0wall web interface rather than during the initial setup of the machine. Through your Ubuntu LAN machine, you should now be able to browse to 192.168.57.101 and began to setup the WAN interface. Assign the M0n0wall WAN interface a static IP address of 192.168.56.102/24 (thus matching the subnet that your Ubuntu WAN machine is on). You have to assign it a default gateway so use 192.168.56.1. Uncheck the “Block private networks” option and save.

Setting up the rules

The M0n0wall now need to be configured with rules that will allow it to operate so that that it will accept all outgoing connections from the LAN side and deny all incoming connections from the WAN unless otherwise stated.

The LAN portion should be already setup as default with it set to “PASS” any traffic from the source “LAN subnet”. Verify this is the case and if for some reason it is not then alter it so that it is.


Firewall: Rules

LAN WAN

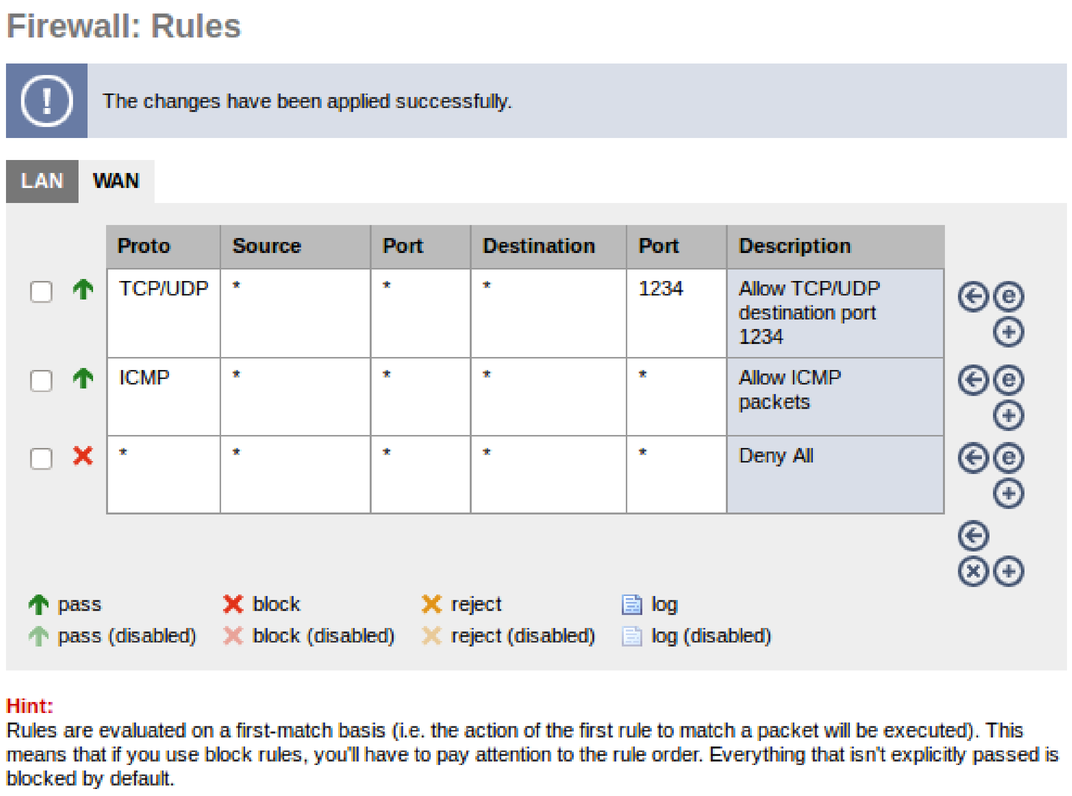
	Proto	Source	Port	Destination	Port	Description
☐ ↑	*	LAN net	*	*	*	Default LAN -> any

↑ pass ✗ block ✗ reject 🗎 log
↑ pass (disabled) ✗ block (disabled) ✗ reject (disabled) 🗎 log (disabled)

Hint:
Rules are evaluated on a first-match basis (i.e. the action of the first rule to match a packet will be executed). This means that if you use block rules, you'll have to pay attention to the rule order. Everything that isn't explicitly passed is blocked by default.

You can now focus on setting up the WAN rules.

The first rule is to add a PASS rule that will allow TCP/UDP connections from any source IP and port to any destination IP with the destination port 1234. Finally you should also add a PASS rule that will allow any ICMP packets to come through. Rules are tested sequentially and when a matching rules is found it is acted on. Thus, in the last rule, you need to DENY ALL connections from all.



Firewall: Rules

The changes have been applied successfully.

LAN WAN

		Proto	Source	Port	Destination	Port	Description
☐	↑	TCP/UDP	*	*	*	1234	Allow TCP/UDP destination port 1234
☐	↑	ICMP	*	*	*	*	Allow ICMP packets
☐	✗	*	*	*	*	*	Deny All

↑ pass	✗ block	✗ reject	🗎 log
↑ pass (disabled)	✗ block (disabled)	✗ reject (disabled)	🗎 log (disabled)

Hint:
Rules are evaluated on a first-match basis (i.e. the action of the first rule to match a packet will be executed). This means that if you use block rules, you'll have to pay attention to the rule order. Everything that isn't explicitly passed is blocked by default.

This is for the ping test later on so you can verify all the connections are working correctly.

The last thing you need to do is boot up the other Ubuntu machine (referred to in this lab as Ubuntu-WAN). Be sure to check that it is running on the network interface vboxnet1 and then boot it up as normal.

Execute the command

```
route -n
```

and note the default gateways of the Ubuntu WAN and Ubuntu LAN.

The default gateway of the Ubuntu-LAN should be 192.168.57.101 which is the LAN interface of the M0n0wall. If it isn't then you need to change the default gateway using `route add default gw 192.168.56.102`

The default gateway of the Ubuntu-WAN should be 192.168.56.102 which is the WAN interface of M0n0wall. If it isn't then you need to run `dhclient` on the Ubuntu-LAN, to get a new IP address from the DHCP running on the m0n0wall LAN interface, or perhaps to reboot the Ubuntu-LAN entirely.

Testing the ability to communicate

In order for this lab to work, you need to ensure that each machine can communicate correctly. Check that you are able to ping the Ubuntu-LAN from Ubuntu-WAN and vice-versa.

Cautions

Be sure that the Ubuntu WAN machine is able to ping the M0n0wall machine. Check the M0n0wall rules and that they are in the correct order, failure to do so will result in potential legitimate traffic being denied by the firewall.

Be sure that when you are setting up the M0n0wall that you uncheck the “Block private networks option”. The reason for this is because you are only replicating a LAN/WAN environment; technically your WAN is under a private IP address. If this is not unchecked then the necessary traffic will be blocked and you will not be able to complete this exercise.

As a last resort, if you do not get bi-directional connectivity: you can set up a NAT rule on Monowall. This NAT rule will take traffic destined for the WAN interface (port 1234) of the M0n0wall and NAT it to the Ubuntu-LAN IP address. NOTE that if you do this, then in the remainder of the lab, whenever the Ubuntu-WAN is trying to connect to the Ubuntu-LAN, it will actually use the IP address of the Monowall's WAN interface and rely on the NAT rule to do the forwarding and translation.

How to do a full packet capture and how to do field-level filtering while capturing

Approach

- You will be using tshark to carry out this goal.

Execution

There are many requirements and different pieces to this particular laboratory so in order to get you started, it is important that you understand the basics for this task. That starts with tshark, what it is, what it does and how it can be an effective tool assisting you with your goals.

Please read about tshark command online. It is essentially the command line version of wireshark.

The tshark command has many different options and flags that can be used to specifically capture and filter what you are looking for. Summarized below are some of the main options that you may use throughout this chapter and the book.

- –i flag is used to represent what interface you want to capture on, which is assumed to be eth0 but you will need to check this according to your own setup.
- –f flag is used as a capture filter such as ICMP packets which is used during the ping commands. Therefore if you only wanted a sensor that would capture ICMP packets then you would use the `–f icmp` option.
- –l flag "*is normally used when piping a live capture to a program or script, so that output for a packet shows up as soon as the packet is seen and dissected, it should work just as well as true line-buffering*"[2] according to the man page of tshark.

An example tshark command is shown below:

```
tshark –l –i eth0 –f icmp
```

This command and its different components will be broken down and discussed in greater detail throughout this chapter as you move through it and see the different circumstances where it is used. At this point all you need to think of it as is the command line version of Wireshark that is very powerful and can be manipulated to provide a very specific type of dataset for you to run a filtering sensor with over your network.

How to build a filtering sensor using nc and sed.

2 man page of nc (Ubuntu).

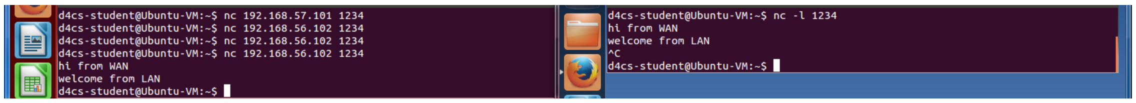
Approach

- Use tshark, nc and sed.

Execution

The first step in building a very simple filtering sensor is to run a nc server on port 1234 on you Ubuntu-LAN machine and connect to it using the client on the Ubuntu12-WAN machine.

You will need to open up two terminals, one on each machine. On the Ubuntu-LAN machine you will need to run the command `nc -l 1234`. This will listen for incoming connections from port 1234 rather than initiate any connections. On your Ubuntu12-WAN machine you should run the command `nc 192.168.56.102 1234` in the terminal. Look carefully at the IP address you just typed in. This is the IP address of the M0n0wall (firewall) machine rather than directly connecting to the LAN machine. Can you think why? It is due to the WAN and LAN machines sitting on separate subnets so they cannot directly communicate with each other. You therefore will need to use the IP address of the firewall and add a port-forwarding NAT rule. This rule will allow "*connections coming from the WAN interface with the protocol TCP/UDP and port 1234 should be translated to 192.168.57.254:1234*". The translated IP address and port refers to the Ubuntu-LAN IP address. Try running the above commands again and see what the results are – it should be successful, as shown below:



```
d4cs-student@Ubuntu-VM:~$ nc 192.168.57.101 1234
d4cs-student@Ubuntu-VM:~$ nc 192.168.56.102 1234
d4cs-student@Ubuntu-VM:~$ nc 192.168.56.102 1234
d4cs-student@Ubuntu-VM:~$ nc 192.168.56.102 1234
hi from WAN
welcome from LAN
d4cs-student@Ubuntu-VM:~$
```

```
d4cs-student@Ubuntu-VM:~$ nc -l 1234
hi from WAN
welcome from LAN
^C
d4cs-student@Ubuntu-VM:~$
```

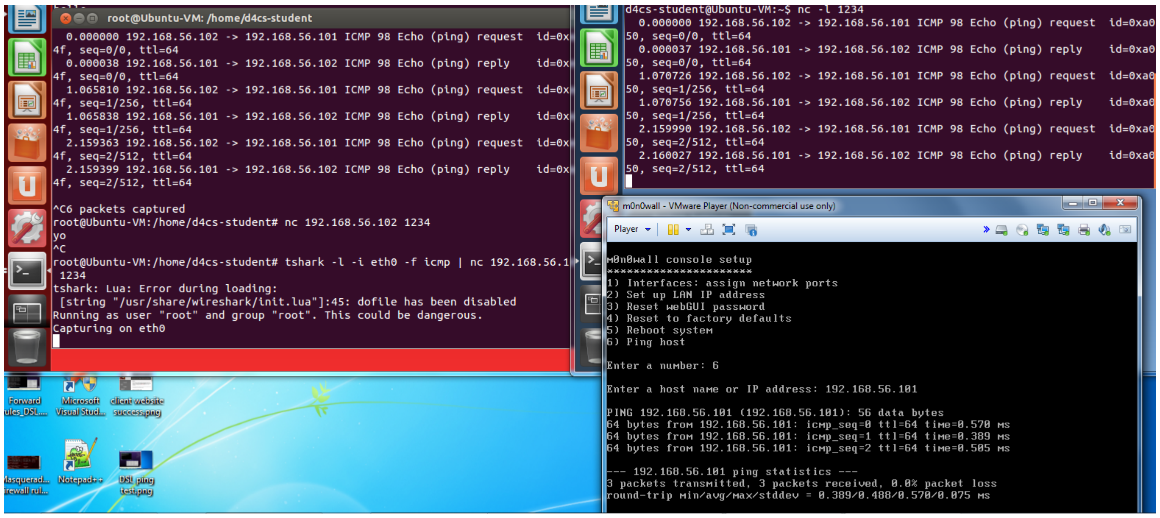
You can now focus on using tshark to capture the necessary traffic and tunnelling it over the network using nc.

With the nc server listening for connections on the Ubuntu-LAN machine, as root you should run the following command on your Ubuntu12-WAN machine:

```
tshark -l -i eth0 -f icmp | nc 192.168.56.102 1234.
```

The –l flag "*is normally used when piping a live capture to a program or script, so that output for a packet shows up as soon as the packet is seen and dissected, it should work just as well as true line-buffering*"[3] according to the man page of tshark. You are using it in your example because you are going to be piping tshark through to nc. The –i flag is used to represent what interface you want to capture on, which is assumed to be eth0 but you will need to check this according to your own setup whilst the –f flag is used, in this instance, as a capture filter for ICMP packets which is used during the ping commands.

Now go to your M0n0wall machine, load up a terminal and try to ping the Ubuntu12-WAN machine. It should be a success. As you can see from the below screenshot, it successfully captured only ICMP packets and using nc and the pipe, transferred them across to the Ubuntu-LAN machine.



```
root@Ubuntu-VM: /home/d4cs-student
  0.000000 192.168.56.102 -> 192.168.56.101 ICMP 98 Echo (ping) request  id=0x
4f, seq=0/0, ttl=64
  0.000038 192.168.56.101 -> 192.168.56.102 ICMP 98 Echo (ping) reply    id=0x
4f, seq=0/0, ttl=64
  1.065810 192.168.56.102 -> 192.168.56.101 ICMP 98 Echo (ping) request  id=0x
4f, seq=1/256, ttl=64
  1.065838 192.168.56.101 -> 192.168.56.102 ICMP 98 Echo (ping) reply    id=0x
4f, seq=1/256, ttl=64
  2.159363 192.168.56.102 -> 192.168.56.101 ICMP 98 Echo (ping) request  id=0x
4f, seq=2/512, ttl=64
  2.159399 192.168.56.101 -> 192.168.56.102 ICMP 98 Echo (ping) reply    id=0x
4f, seq=2/512, ttl=64

^C6 packets captured
root@Ubuntu-VM:/home/d4cs-student# nc 192.168.56.102 1234
yo
^C
root@Ubuntu-VM:/home/d4cs-student# tshark -l -i eth0 -f icmp | nc 192.168.56.1
 1234
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:45: dofile has been disabled
Running as user "root" and group "root". This could be dangerous.
Capturing on eth0
```

```
d4cs-student@Ubuntu-VM:~$ nc -l 1234
  0.000000 192.168.56.102 -> 192.168.56.101 ICMP 98 Echo (ping) request  id=0xa0
50, seq=0/0, ttl=64
  0.000037 192.168.56.101 -> 192.168.56.102 ICMP 98 Echo (ping) reply    id=0xa0
50, seq=0/0, ttl=64
  1.070726 192.168.56.102 -> 192.168.56.101 ICMP 98 Echo (ping) request  id=0xa0
50, seq=1/256, ttl=64
  1.070756 192.168.56.101 -> 192.168.56.102 ICMP 98 Echo (ping) reply    id=0xa0
50, seq=1/256, ttl=64
  2.159990 192.168.56.102 -> 192.168.56.101 ICMP 98 Echo (ping) request  id=0xa0
50, seq=2/512, ttl=64
  2.160027 192.168.56.101 -> 192.168.56.102 ICMP 98 Echo (ping) reply    id=0xa0
50, seq=2/512, ttl=64
```

```
m0n0wall - VMware Player (Non-commercial use only)

m0n0wall console setup
**********************
1) Interfaces: assign network ports
2) Set up LAN IP address
3) Reset webGUI password
4) Reset to factory defaults
5) Reboot system
6) Ping host

Enter a number: 6

Enter a host name or IP address: 192.168.56.101

PING 192.168.56.101 (192.168.56.101): 56 data bytes
64 bytes from 192.168.56.101: icmp_seq=0 ttl=64 time=0.570 ms
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.389 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.505 ms

--- 192.168.56.101 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.389/0.488/0.570/0.075 ms
```

ARP Filtering

At this point you have a very basic filtering sensor using tshark but it is not very useful in a real world environment. You are now going to begin to filter ARP data in a manner that only captures what you are really interested in. This could be useful to detect such attacks as ARP poisoning. There are going to be five fields that you will be interested, they are:

1. opcode

3 man page of nc (Ubuntu).

2. source MAC
3. source IP
4. Destination MAC
5. Destination IP

Initially, try to look through the man page for tshark and see if you can come up with the command that you will use that will allow you to capture those five fields and store them to a file. To help you with this task, a helpful clue is to look at the corresponding display filters to the type of data that you want to capture. If you are having trouble then the answer is below.

```
tshark –i eth0 –f arp –T fields –E separator=/s –E
aggregator=, -E quote=s –e frame.time –e arp.opcode –e
eth.src –e arp.src.proto_ipv4 –e eth.dst –e
arp.dst.proto_ipv4 > ARP_output.csv
```

Lets break down what each part of the command does so you fully understand what you are about to execute.

Command	**Meaning**
`-i eth0`	Use tshark to capture traffic on the eth0 interface.
`-f arp`	Filter the capture so that it only captures ARP traffic.
`-T fields`	Separate the data into fields and place into a csv file
`-E separator=/s`	Use the single space as the separator, in tandem with the –T flag.
`-E aggregator=,`	After each field add a comma at the end.
`-E quote=s`	Surround the contents of each field response in single quotes.
`-e`	Specify which type of fields you want to capture in your command. In this instance, the five that were previously mentioned.
`> ARP_output.csv`	Send the output into a file called ARP_output.csv.

Now that you know the breakdown of the command and precisely what it does, run it on the Ubuntu12-WAN machine for a while. Once you think you have enough, end the command and look at the contents of the file `ARP_output.txt`. Your file should look like a typical csv file along with the contents of the 5 ARP fields that you wanted to capture.

sed manipulation

The next is to convert that current data into a format that can be used to input it into a database. This is where the sed command is incredibly useful.

Please read about the sed command online, focussing on the search and replace directives and how regular expressions that can be used to match substrings and replace them with other strings.

The command to correctly form SQL statements using the sed command is shown below:

```
sed –e “s/, //” –e “s/://g” –e “s/\.[0-9]*′/′/”
ARP_output.csv | sed –e ‘s/Jan /01/I’ –e ‘s/Feb /02/I’ –e
‘s/Mar /03/I’ –e ‘s/Apr /04/I’ –e ‘s/May /05/I’ –e ‘s/Jun /
06/I’ –e ‘s/Jul /07/I’ –e ‘s/Aug /08/I’ –e ‘s/Sep /09/I’ –e
‘s/Oct /10/I’ –e ‘s/Nov /11/I’ –e ‘s/Dec /12/I’ | sed –e ‘s/
//’ | sed –e ‘s/^/INSERT INTO
arpdata(ts,op,srcmac,srcip,dstmac,dstmac,dstip) VALUES(/’ |
sed –e “s/′ ‘/′,′/g” | sed –e ‘s/$/);/’
```

The question is, what does such a big command actually mean. This is broken down and explained below in smaller pieces:

```
sed   –e   “s/,   //”   –e   “s/://g”   –e   “s/\.[0-9]*′/′/”
ARP_output.csv |
```

The first `sed –e “s/, //”` will remove the first comma from the date between the day and year. The next statement `–e “s/://g”` removes all the colons (:) from the data because the g is at the end is a global statement so it removes all instances. The last part of the above statement `–e “s/\.[0-9]*′/′/” ARP_output.csv |` will look for the first period (.) followed by zero or more numbers until it hits the next apostrophe (‘) and delete that section. This removes the numbers after the second section in the timestamp. Finally the `|` will pipe the resulting data set through to the next set of commands.

```
sed –e ‘s/Jan /01/I’ –e ‘s/Feb /02/I’ –e ‘s/Mar /03/I’ -e
‘s/Apr /04/I’ –e ‘s/May /05/I’ –e ‘s/Jun /06/I’ –e ‘s/Jul /
07/I’ –e ‘s/Aug /08/I’ –e ‘s/Sep /09/I’ –e ‘s/Oct /10/I’ –e
‘s/Nov /11/I’ –e ‘s/Dec /12/I’ |
```

The section above replaces the three monthly letters and the following space for each possible month and replaces it with its corresponding two-digit month number instead. The `/I` at the end of each statement indicates that it is case insensitive, this way it does not matter if it was ever reported in uppercase, lowercase or any combination of the two.

```
sed –e ‘s/ //’ |
```

This section will remove the first space that it finds, which in this case is the space between the date and the time entry.

```
sed –e ‘s/^/INSERT INTO
arpdata(ts,op,srcmac,srcip,dstmac,dstmac,dstip) VALUES(/’ |
```

This will insert the INSERT INTO statement at the beginning of the file because it matches the (^) character. In sed the `^` signifies the beginning of the line.

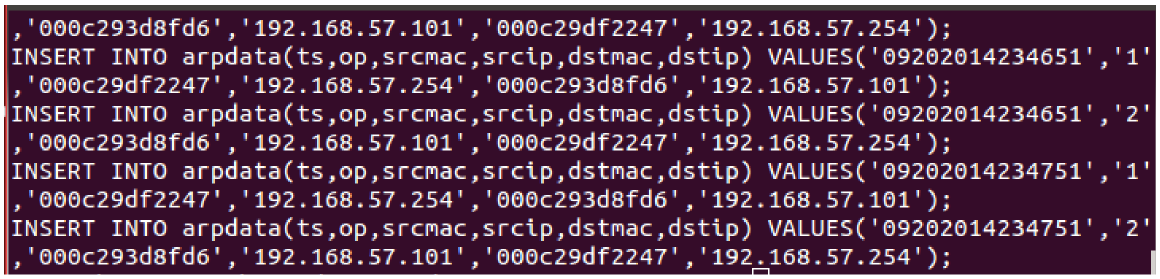
```
sed –e “s/’ ‘/’,’/g” |
```

The code above will globally replace all current spaces with a comma, as required for the SQL Syntax.

```
sed –e ‘s/$/);/’
```

Finally, this last statement will add `);` to the end of the data because it matches on $ which in sed means to match at the end of the line.

You should now be ready to covert your command into properly formed SQL statements. Try running the command and send the output to another file called `SQL_Data.csv`. Please see the screenshot below, which is an indication that the outcome worked as desired. You should see something similar:



```
,'000c293d8fd6','192.168.57.101','000c29df2247','192.168.57.254');
INSERT INTO arpdata(ts,op,srcmac,srcip,dstmac,dstip) VALUES('09202014234651','1'
,'000c29df2247','192.168.57.254','000c293d8fd6','192.168.57.101');
INSERT INTO arpdata(ts,op,srcmac,srcip,dstmac,dstip) VALUES('09202014234651','2'
,'000c293d8fd6','192.168.57.101','000c29df2247','192.168.57.254');
INSERT INTO arpdata(ts,op,srcmac,srcip,dstmac,dstip) VALUES('09202014234751','1'
,'000c29df2247','192.168.57.254','000c293d8fd6','192.168.57.101');
INSERT INTO arpdata(ts,op,srcmac,srcip,dstmac,dstip) VALUES('09202014234751','2'
,'000c293d8fd6','192.168.57.101','000c29df2247','192.168.57.254');
```

-

How to build a persistence service to store packets.

<u>Approach</u>

- MySQL

<u>Execution</u>

At this point you should have successfully created a csv file that has a list of SQL commands that you can use. The next step is to utilize that SQL server that you should have setup during the prerequisites stage. If you missed this step then that’s ok, just go back and follow the steps relating to the SQL server setup.

With your SQL server now up and running you can begin the process of creating the necessary databases and tables in order to input data into them. You can do this by creating, running and executing a simple shell script that will execute valid SQL commands when run. The script should look something like the below:

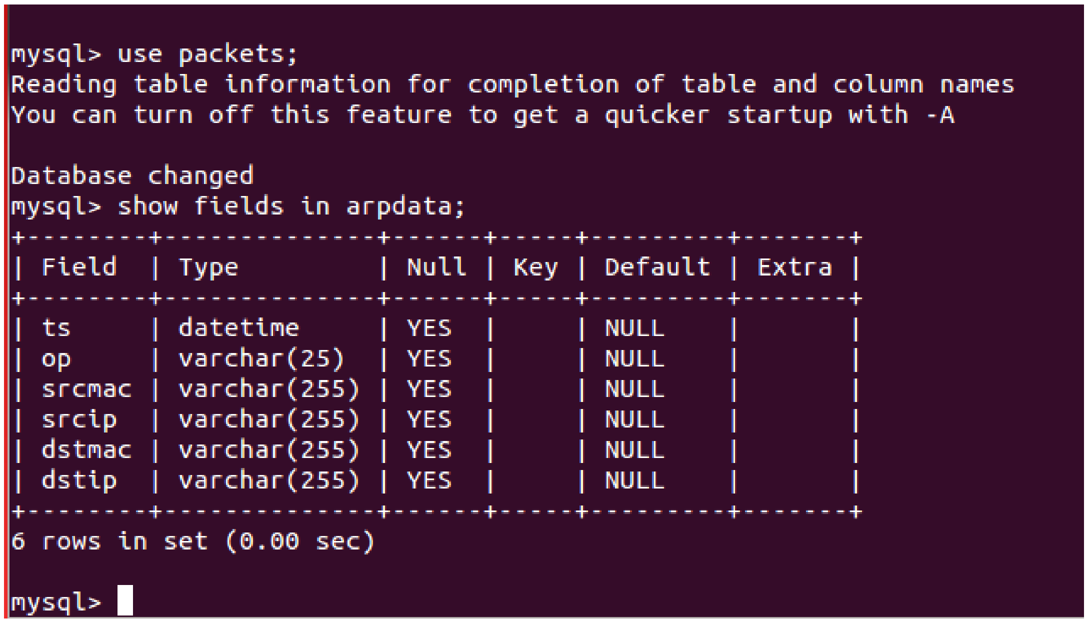
```
#!/bin/bash
#Create the database and tables
mysql --user=root --password=[your SQL password] <<EOF
CREATE DATABASE IF NOT EXISTS packets;
USE packets;
CREATE TABLE IF NOT EXISTS arpdata (
ts DATETIME,
op VARCHAR(25),
srcmac VARCHAR(255),
srcip VARCHAR(255),
dstmac VARCHAR(255),
dstip VARCHAR(255)
);
EOF
```

Type the above into a text file and save it with the filename `createddb.sh`. Let's break it down and understand what it is that the shell script will be doing. The first line is to tell the computer that it is a bash shell script. The second line attempts to log into your MySQL server environment by providing the correct credentials. The following line is a comment to inform the reader of the script of what its intended purposes is. The next two lines try and use the database called `packets` and if it doesn't exist then create it. The next line creates a table called `arpdata`. The following lines then create the various columns including `ts`, `op`, `srcmac`, `srcip`, `dstmac` and `dstip`. These columns will hold the data that the sensor filter captures.

Remember that before you can execute the script, you need to assign it the correct permissions. use the following command:

```
#chmod 755 createdb.sh
```

This will change the permissions of the file so that it can be executed. You should now be able to run the script and it should run the SQL commands, as shown below:



```
mysql> use packets;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show fields in arpdata;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| ts     | datetime     | YES  |     | NULL    |       |
| op     | varchar(25)  | YES  |     | NULL    |       |
| srcmac | varchar(255) | YES  |     | NULL    |       |
| srcip  | varchar(255) | YES  |     | NULL    |       |
| dstmac | varchar(255) | YES  |     | NULL    |       |
| dstip  | varchar(255) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> 
```

All of the above work was just preparation for the next stage, which is to combine it all together, including the previous task, and be in a situation where a script will run that will create the database and table for SQL and read in valid SQL INSERT statements from your filter. You can attempt to do this yourself but an example is shown below:

```
#!/bin/bash
#Create the database and tables
echo 'mysql --user=root --password=[your SQL password]
<<EOF'
echo 'CREATE DATABASE IF NOT EXISTS packets;'
echo 'USE packets;'
echo 'CREATE TABLE IF NOT EXISTS arpdata ('
echo -e '\t ts DATETIME,'
echo -e '\t op VARCHAR(25),'
echo -e '\t srcmac VARCHAR(255),'
echo -e '\t srcip VARCHAR(255),'
echo -e '\t dstmac VARCHAR(255),'
echo -e '\t dstip VARCHAR(255)'
echo ');'
#Run tshark and send the output through the filter
tshark -l -i eth0 -f arp -T fields -E separator=/s -E
aggregator=, -E quote=s -e frame.time -e arp.opcode -e
eth.src -e arp.src.proto_ipv4 -e eth.dst -e
arp.dst.proto_ipv4 2>/dev/null | sed -u -e "s/, //" -e
"s/://g" -e "s/\.[0-9]*'/'/" | sed -u -e 's/Jan /01/I' -e
```

```
's/Feb /02/I' -e 's/Mar /03/I' -e 's/Apr /04/I' -e 's/May /
05/I' -e 's/Jun /06/I' -e 's/Jul /07/I' -e 's/Aug /08/I' -e
's/Sep /09/I' -e 's/Oct /10/I' -e 's/Nov /11/I' -e 's/Dec /
12/I' | sed -u -e 's/ //' | sed -u -e 's/^/INSERT INTO
arpdata(ts,op,srcmac,srcip,dstmac,dstip) VALUES(/' | sed -u
-e "s/' '/','/g" | sed -u -e 's/$/);/'
echo 'EOF'
```

You should notice that there are a few subtle differences to the tshark command. The `–l` option on the tshark command is meant for live captures. There has also been the –u command at the beginning of each sed command as this will "*load minimal amounts of data from the input files and flush the output buffers more often*"[4] according to the man sed page. This is required when piping through live data. The last step was to remove the error messages that tshark output from your INSERT INTO statements that it had incorrectly captured. This is done by adding `2>/dev/null`, according to the GNU Bash Manual[5]. This would redirect all errors to `/dev/null` which is a place where data is simply discarded on the Linux system.

Before you finally put everything together you need to ensure that MySQL server on the Ubuntu12-WAN is running correctly. Run the following command to ensure the server is running:

```
#sudo mysql restart
```

Check that the service is running correctly by issuing the following command:

```
#sudo service mysql status
```

It should return with a status of running. If so, you are all set.

Putting it all together you now need to augment your solution so that you are effectively piping packet captures from the Ubuntu-LAN using nc to the Ubuntu12-WAN machine, where you are inserting the captured data into a mysql database.

4 man page for sed
5 http://www.gnu.org/software/bash/manual/bashref.html#Pipelines

You will need to run your new script on the LAN machine and pipe it through nc 192.168.56.101 1234.

Extra Credit: Build a basic GUI interface to visualize captured packet data.

Set up your LAN Ubuntu Apache and extend it to be able to serve PHP scripts and connect to the MySQL server (This is called a LAMP setup, that is... Linux Apache MySQL PHP).

Read about: How to write an HTML web page; how to write a PHP program; how to write a PHP program that connects to a local MySQL database.

Now: Write a PHP program which reads the packet data from the database and shows it in a webpage as an HTML table. Put this PHP program into /var/www/ and then test that you can examine stored packets using your web browser.