Introduction to Computer Networking
Netkit lab – Application layer

Emeline Marechal Simon Liénardy

Academic year 2021 – 2022

Introduction

This lab is an introduction to Netkit. Netkit enables to simulate a network by connecting several virtual machines. During this session, you will be introduced to some very useful tools such as `tcpdump` and `wireshark`. The description of this laboratory introduces tools that will be used in future labs.

Contents

1 You must be new here – Introduction to netkit

First of all, download the sources from eCampus and extract them in the directory of your choice. The following will name the extracted directory "labs directory".

1.1 A first lab

1. First, go to the `firstLab` directory. You can use your mouse but you will look more serious and clever using a terminal (Notes after `#` are comments and are not interpreted by the terminal).

```
$ cd path # target path
$ cd ..   # Go to the parent directory
```

   The command `cd` stands for **C**hange **D**irectory. The target path is composed like this: `dir/subdir/target` where `target` is a directory contained in `subdir` whose *parent* is `dir`.

2. Once you have reached `first_lab` directory, please type the following in order to start your first lab:

   ```
   $ lstart
   ```

   You should see that a new terminal popped up, in which you can enter commands that will be executed in a vm named `vm`.

   A quick look at: Guest and Host systems

   - The **host** machine is the machine on which the VMs are running.
   - The **guest** machines are the VMs.

3. In `vm`, type the following command:

   ```
   vm$ ls /hosthome
   ```

   `ls` command will **L**i**S**t the content of the `home` directory of the host. You can access the host file hierarchy from the guest. This will be more than useful in the next labs.

4. A lab with a single vm is not very interesting since we want to experiment networking. To complete our quick introduction to `netkit`, we can create a new vm.

   ```
   host$ vstart pc
   ```

   This will create a new vm named `pc`.

   Try the following commands. What can you observe? All these problems will be solved in the next section.

   ```
   pc$ ping vm
   ```

   ```
   vm$ ping pc
   ```

   A quick look at: `ping`

   This tool is mainly used to test the reachability of a given device, and to measure the delay on a network, meaning the time between the message sent and its reply. It is based on two types of ICMP messages: *echo request* and *echo reply.* Syntax:

   ```
   ping dest
   ```

   Where `dest` can be either a name or an IP address.
   `ping` continuously sends probes. To stop it, press `Ctrl` + `C`.

5. In `firstLab` directory, you can see files named `vm.disk` and `pc.disk`, these are used to preserve file system contents across reboots of the virtual machine. There is one file system (fs) copy and `*.disk` files store differences between them and the original fs. After you stopped your lab, you can erase them.

   To sum up, table 1 lists common `netkit` commands:

Command	Usage
`vstart name`	Start a vm called `name`
`vhalt name`	Halt a vm called `name`
`vcrash name`	Crash a vm called `name`. This will erase `name.disk` file.
`vlist`	List the running VMs
`lstart`	Start a lab in current directory (`lab.conf` file must exist)
`lhalt`	Halt the lab launched from current directory
`lcrash`	Crash the lab launched from current directory, `*.disk` files are erased.

Table 1: Netkit commands

1.2 A new hope

This part uses the lab contained in directory `newHope`.

1. Start the lab
2. Try to `ping` the first VM from the second and vice versa. Now, it's working. In the result line, you read `min/avg/max/mdev`. What do `avg` and `mdev` stand for? Why is it important to know these values?
3. Open `*.startup` files and `lab.conf` file. They contain useful command to connect VM in netkit. In `lab.conf`, you can see commands like:

```
vm1[0]=A
vm2[0]=A
```

The synthax of the command is: `vm_name[iface_number]=C_D`, where `vm_name` is the name of a VM of the lab, `iface_number` is an interface number, `C_D` is a collision domain name (here represented by a capital letter). The two commands above say that `vm1` and `vm2` are connected together by the same collision domain and communicate both through their ether0 interfaces (0 refers to ether0 interface).

A quick look at: Collision Domain

A collision domain can be seen as a sort of virtual hub. Thus, attaching interfaces of different virtual machines to the same collision domain allows them to exchange network traffic[a].

[a] From `vstart` manpage

In `*.startup` files, you can see calls to the `ifconfig` program that helps configure interfaces, attribute them IP addresses (more details in a few sessions). We can represent the *topology* of the lab and sum up the information contained in configuration files by drawing Figure 1.

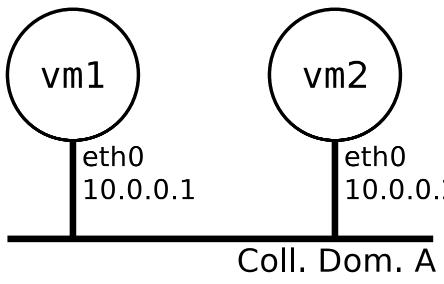

Figure 1: Second Lab topology

2 Experiment with HTTP

In this part, we will download from the server the file named "`download.txt`" and we will look at the packets exchanged between the client and the server. The global procedure to do this is the following:

- Launch `tcpdump` on the client to capture the traffic.
- Download the file from the server.
- Kill `tcpdump`.

1. Open `http` directory. Draw the topology (as in Figure 1) on the basis of the configuration files.
2. On server side, type:

```
server$ python -m SimpleHTTPServer 80
```

A quick look at: `python` running module as a script

`python` command launches python interpreter. The `-m` option will execute its module parameter (here `SimpleHTTPServer` as a script). `80` is here the parameter given to the `SimpleHTTPServer` and represents the port number (80 is the reserved port number for http traffic), and 443 is the reserved port number of https traffic.

Note that you can use port 80 here because you are the root user[1]. The tree of the server will be rooted in the current directory, e.g.:
`server/` is the URL for current directory;
`server/foo/bar.html` is the URL for the file `bar.html` located in directory `./foo`.

3. Start sniffing the network with `tcpdump` on the client side. Save the dump in a file called "`analysis.pcap`" (for example), located in the host system.

[1]If you have not the rights to use port 80, you can use 8080 for http

You can do that as a background job with the following command:

```
client$ tcpdump -i eth0 'tcp' -s 0 -w /hosthome/capture.pcap &
```

`-s 0` enables to capture the entire packets (without truncating them). Do not forget to type `&` at the end of the line to execute the command in a background job. The PID number of the job is then displayed.

4. On the client side, you can now download a file from the server using `wget`. A file named "`download.txt`" is available on the server for a download. It is your task to find the proper URL.

   As an **example**, if we wanted the file `bar.html` located in directory `./foo` from the server `server`, we would invoke:

   ```
   client$ wget server/foo/bar.html
   ```

5. When the file is totally downloaded, don't forget to stop `tcpdump` using:

   ```
   client$ kill PID
   ```

   Where PID was the PID of tcpdump that was displayed just after this program was invoked.

6. It is now time to look at the packets that were exchanged during the download. From host system, open `capture.pcap` with `wireshark`. Look at section **??** to understand the user interface. You can see all the packets that were exchanged. Two of them are labeled as HTTP: a GET and a response. But there are more than 2 packets. Why?

7. Look at the encapsulation of the packets. Which protocols are used? What are their addresses?

8. What can we find in the HTTP response header? What are the headers present in the response? What is the response code?

9. Consider this sentence: "HTTP Data are sent in clear". Explain. What can you conclude about your privacy?

10. Try the following situations . You can dump the packets exchanged and analyze them with `wireshark`. Remember the global procedure presented above.

    - a misspelled URL;
    - try another destination port (How to specify a port in a URL?);
    - try to connect to the server using the HTTPS protocol. What happens? And if the server is listening on port 443?

3 DNS

Remark: In this lab, the DNS configuration is overly simplified.

1. In previous sections, we focused on TCP traffic that reached the client. We used a URL to access the server remotely. How does the client know how to reach the server? In particular, how does it know its IP address?

2. Capture DNS traffic (you must change the `tcpdump` capture option)[2]. Compare it with the HTTP over TCP traffic we saw earlier. A machine asks for a reverse DNS lookup. Which one?

3. How is the `dns` machine configured? In lab `newHope`, there was no DNS server. How did you manage to address machines by their names?

4. Look at `/etc/bind/` directory in `dns` source files. The `named.conf` file is the main configuration file. In this file, you see that zone `montefiore` was defined and information about this zone is stored in `db.montefiore` file. In this last file, there are resource records. The file `db.10` that is used for reverse lookup (translate an IP into a name).

5. Stop the `dns` machine. How can the client contact the server? Try your solution.

6. Why would a server ask for the translation of an IP into a name (also called reverse DNS)? *Hint: in the real world, this is often performed by mail servers.*

A quick look at: Resource record

The format of a resource record is:

```
NAME CLASS TYPE RECORD_DATA
```

The `NAME` is the record owner[a]. The class is usually `IN`. The type present in the file are described below. The `RECORD_DATA` depends on the record type.
There are 3 types of record in the db file of the lab:

- `SOA`: contains the name of the primary master server for the zone, an address of contact (first dot has to be replaced by `@` and other configuration values that are used for master/slave server config and are useless for this lab);
- `A`: contains an IPv4 address;
- `NS`: the authoritative name server for the domain.
- `PTR`: a domain name pointer (for reverse DNS lookup. In the lab, `NAME` is the last byte of the IP address and `RECORD_DATA` contains host name, thus the "inverse" information contained in the A records.).

[a]the name of the node to which this resource record pertains [rfc 1035]

[2]You can capture all the packets to see the DNS probe in the context of the HTTP messages