RES212 LAB #2
NETFILTER/IPTABLES FIREWALL

The goal of this lab is to let you become acquainted with the design, configuration and testing of firewalls. Given the limited amount of time, we limitedly focus on packet-level and circuit-level gateways, without advanced application-layer proxy functionalities. In this Lab, we adopt an opposite viewpoint with respect to the previous one and put more emphasis on the networking environment (as opposite to the cryptographic aspects, which would not be well captured by the circuit gateway anyway, and that Lab #1 already dealt with).

The Lab is divided in 4 main phases:

1. ***Setting up the emulated network (Netkit UML)***
2. ***Defining and implementing the firewall policies***
3. ***Functional tests and performance benchmark***
4. ***Refine the firewall configuration***

The Lab is carried on using a Netkit emulator. Netkit leverages User Mode Linux (UML) to provide access to terminals of different entities (server, clients, router, firewalls), of which you have full root access. Whereas Netkit emulation does not provide faithful performance results *(due to the UML overhead) however it provides accurate* functional results, which are essential for the Firewall lab. *Note that, once you launch Netkit, you are already* root *of all the machine.*

1. Setting up the emulated network (Netkit UML)

Note: this part helps you in setting up your Netkit environment. It gives you very basic hints but assume you are familiar with Linux. The use of the same VM image ensures that everybody has an (almost) consistent setup.

Setting up netkit is pretty simple: all you have to do is to download the modified Netkit image (`netkit-RES212.tgz`) from the RES212 webpage, extract and configure the required environmental variable (`./check_configuration.sh`)

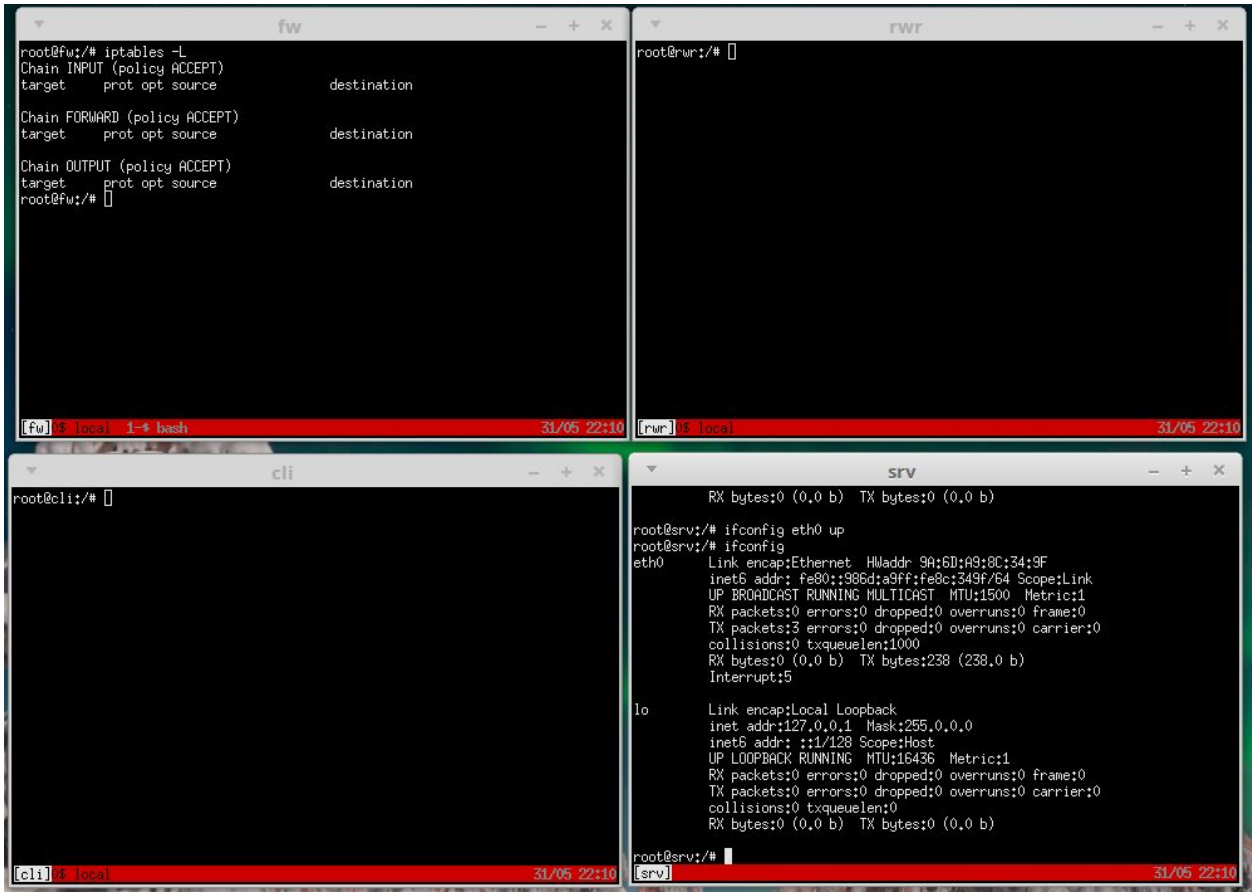
```
user@c126-1 /tmp $ tar -xzvf netkit-RES212.tgz
user@c126-1 /tmp $ cd netkit-RES212; ./check_configuration.sh
[...]
user@c126-1 /tmp $  export NETKIT_HOME=/tmp/netkit-RES212
user@c126-1 /tmp $  export PATH=/tmp/netkit-RES212/bin:$PATH
```

Now, a temporary netkit installation is available in the host. You can execute a lab by downloading and extracting the Lab02 (`RES212-Lab02.tgz`) from the RES212 webpage, then using Netkit `lstart` from the Lab directory:

```
user@c126-1 ~ $ tar -xzvf RES212-Lab02.tgz
user@c126-1 ~ $ cd RES212-Lab02
user@c126-1 RES212-Lab02 $ lstart -s -f
======================== Starting lab ============================
Lab directory: /home/drossi/Dropbox/RES212/lab2
Version:       0.1
Author:        Dario Rossi
Email:         dario [dot] rossi [at] enst [dot] fr
Web:           res212.telecom-paristech.fr
Description:   RES212 - Firewall lab
==================================================================
Starting "fw" with options "-q --eth0 "Internet" --eth1 "LAN" --eth2 "DMZ"
--hostlab=/home/drossi/Dropbox/RES212/lab2
--hostwd=/home/drossi/Dropbox/RES212/lab2"...
...

The lab has been started.
==================================================================
```

Organize the desktop as you wish: an example with 4 tiled terminal follows:



We have configured the terminal to start with `screen`, which is useful to have *many virtual terminals for the same machine* (see the 0-local and 1-bash in the top left corner): to create and loop among several terminals in the same machine use the following key shortcuts (see ``man screen'' for more)

- Ctrl+a c
  create a new new window
- Ctrl+a n
  move to the next window

Netkit provides a handy access to the host directories, in particular

- `/hosthome`
  allows you to access all your files (i.e., those of your TPT DSI account)
- `/hostlab`
  `Provides` access the Lab folder you are running from; useful if you want to save results or commands that are inherent the lab.

To stop a Netkit lab you can either :

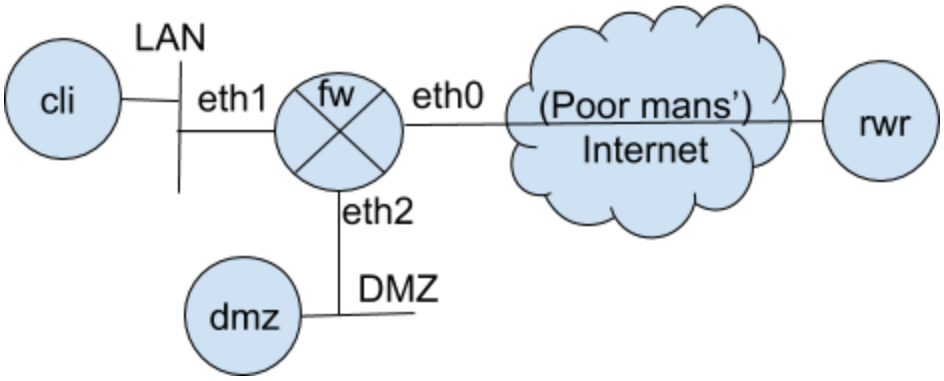
- `lcrash` from the Lab directory:
  Crash abort machines without saving any state (e.g., configuration, programs installed, etc.). This is handy when you messed the configuration and want to restart from scratch, or try a different firewall design.
- `lhalt` from the Lab directory:
  gracefully send shutdown signals and modifies changes to the disk (i.e., you can find system-wide configuration and programs installed from previous session on the next boot). This is handy if you want to resume the Lab at a later time (beware that *system-wide configuration* is different from *session state:* in other words, editing a conf file in `/etc/` is different than running a command in the terminal, as the result of the latter is forgotten after a reboot.

These basic infos should be enough for the lab, but visit Netkit http://wiki.netkit.org/ if you want to know more.

This is the first edition of this Lab. The lab has been tested by PhDs, so we have ideas on its execution time for graduate students. However, execution time for undergraduate students is totally unknown. It would be very valuable if you could note down the time you arrive at the end of each section and share it in this form

2. Defining and implementing the firewall policies

Goal: understand the basic Netfilter/iptables policies, hooks, chains and actions. Flush and load firewall configuration. (Optionally: configure the network)



The Netkit lab is configured to keep the number of hosts to the minimum possible, while still allowing for some simple yet insightful exercises to be performed on this setup: namely, we use a three-legged firewall configuration with a DMZ and a LAN. DMZ and LAN have private address so that the FW has to put NAT in place (although of different types for different purposes) for the DMZ and LAN.

Simplification #1: Given that we use a software router, all operations including forwarding packet-level filtering and circuit-level gateway are colocated at fw (i.e., we do not have a standalone router)

Simplification #2: The internet is poorly represented as fw and rwr are directly physically connected and thus are in the same collision range (this may create confusion for the addressing).

Simplification #3: For the sake of simplicity, instead of adding an additional terminal for an additional machine used for penetration testing, we assume that roadwarrior rwr can be either:

- Used for legitimate activities, such as accessing DMZ servers, or accessing the network through a configured VPN
- Used for penetration testing or abused for illegal activities, in an attempt to penetrate the network we aim at protecting

Based on the business practices of the company, its Chief Security Officer (CSO) has defined the following security high-level policies and requirements that is your job to define at a fine-grain and implement. The following is your assignment for part 2 of the lab.

A) Use a whitelisting policy
B) Forward traffic between DMZ and the Internet
C) Forward traffic between LAN and DMZ only if initiated from the LAN
D) Forward SSH, Web and DNS traffic between LAN and Internet only if initiated from LAN
E) Drop all traffic initiated from Internet to the FW router except ICMP echo request/reply
F) Allow traffic from FW to anywhere, but only allow SSH access from the LAN

We know that the CSO will refine them later (section 4), but for the time being we are interested in learning how to implementing these policies (e.g., iptables syntax), save reload and swap firewall configurations (e.g, management of iptables policies).

2.1 Configure the network (optional)

First of all, you need to configure the network. If you want to skip this test, we have provided an already configured version of the lab: if you look carefully, you should already have extracted it! If you do or even if you don't do 2.1, time it please!

If you prefer to get your hands dirty or don't like the IP addressing already provided, you can configure the network as you wish starting from scratch bringing up the interfaces and configuring the necessary (static) routes. Clearly, you should not forget to:

- Assign addresses (and netmask) and bring interfaces up
- Add network routes to directly connected interfaces (with the right netmask)
- Add a default gateway route
- Enable IP forwarding on the firewall
- Configure the name resolution (statically in /etc/hosts)

A quick and dirt way to check that you got the configuration correct, is to test reachability of any interface in the network, from any host in the lab. Example from roadwarrior:

```
root@rwr:/# for k in fw0 fw1 fw2 cli dmz; do ping -c1 $k; done
```

2.2 Check the default policies (A) and adopt good practices

Now, from the firewall terminal, check the default policies:

```
root@fw:/# iptables -L
```

Comment your output:
Is the firewall configured for whitelisting or blacklisting?
Do you consider it a safe choice ?

To check if the default policies can leak any useful information (or worse, open a surface for attack), from the rwr terminal, check if you see any open service:

```
root@fw:/# nmap fw
```

System security related comment:
given requirement **F),** shouldn't the sysadmin modify the default `/etc/ssh/ssh_config` on fw ? How would you fix it ?
Remember security needs an holistic view. *Network security* alone (and packet filtering as in this lab) is just one of the tools of the trade.

While of course you should use the terminal to familiarize with iptables syntax, it is a good idea to start on good practices. Thus, save the iptables configuration with the following command: this will save it into the lab directory in the host, so the configuration file will persist after you stop/crash the lab.

```
root@fw:/# iptables-save > /hostlab/firewall.conf
```

Now, check the content of the file: you can check it using a texteditor from the fw:

```
root@fw:/# pico /hostlab/firewall.conf
```

or since the file is anyway stored on the disk of the host machine, you can also use a GUI editor if you prefer:

```
user@c126-1 ~/path/to/your/lab $ gedit firewall.conf
```

No matter how you are accessing this file, you see that the content is messy, right? So please take a minute (or two) to cleanup the text file. Comments are allowed and suggested, so that for instance you make your set of instructions much more readable (for your CSO, fellow colleagues and for yourself):

```
root@fw:/hostlab# cat firewall.conf
# RES212 - lab02 configuration
# requirement A) whitelisting
*filter
:OUTPUT DROP
:INPUT DROP
:FORWARD DROP
COMMIT
```

This way, it is pretty simple to save, restore and check clean configuration rules:

```
root@fw:/hostlab# iptables-restore < firewall.conf

root@fw:/hostlab# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

You can test this rule to be in effect by trying to, e.g., ping the DMZ from the LAN or the Internet. A handy command for the debut is to have iptables to continuously report which rule was hit when you issue some active probing (e.g., ping). The `watch` command is a neat way to achieve this: keep a `screen` terminal open with the following command and quickly alternate between changing configuration and seeing its impact with the <Ctrl a, n> `screen` keybinding.

```
root@fw:/hostlab# watch iptables -v -n -L
rules:Every 2.0s: iptables -n -v -L               Fri Jun  1 10:10:55 2018

Chain INPUT (policy DROP 145 packets, 12180 bytes)
 pkts bytes target     prot opt in     out     source destination

Chain FORWARD (policy DROP 163 packets, 13444 bytes)
 pkts bytes target     prot opt in     out     source destination

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source destination
```

2.3 Forward traffic between DMZ and the Internet (B)

Now, you should allow traffic to enter the DMZ. Restrict traffic forwarded between DMZ and the Internet to traffic of a set of well-known services identified by their port number. In particular, the DMZ runs a public Web server (HTTP and HTTPS) and an iperf server.

- If the DMZ had a set of public addresses, which chain should have the firewall used ? input forward or drop ?
- Think about the input and output interfaces: does explicitly specifying the interface gives additional security ?

Whereas in this Lab the rwr is just another client connected to the firewall (and thus can access the dmz server despite it has a private address in the 10/8 block), if we added Internet routers, then the private 10.2.0.1 address would never be reachable from the Internet.

Setup a DNAT where D stands for Destination in Linux terminology, and that would be called a Static NAT under Cisco terminology.

- Which chain should the firewall use ?

2.4 Forward traffic between LAN and DMZ only if initiated from the LAN (C)

To do so, you need the conntrack module, and specify the state:

Think about TCP connection opening: a NEW connection from the LAN will need to send a TCP packet with a SYN flag set to some server in the DMZ

- If such a server is listening in the DMZ, to complete the handshake, it will have to reply with a SYN-ACK packet that is RELATED to the original SYN
- If such a server is not listening, the network stack will have to send an ICMP port unreachable message to avoid the TCP client waits indefinitively for the reply (is not indefinite, but very long: you can time it if you want)

2.5 Forward SSH, Web and DNS traffic between LAN and Internet only if initiated from LAN (D)

Now you need to further specify the ports and setup NAT for LAN clients

- Use well-know ports for destination; Does it make sense to also restrict the portrange for the source? If so, how? If not, why?
- In this case, to let your NAT clients having private address to be reachable from the outside, you should setup a Source NAT (SNAT) in Linux terminology, that would be called Dynamic NAT in Cisco terminology

2.6 Drop all traffic initiated from Internet to the FW router except ICMP echo request/reply (E)

- Think about the tradeoff between silence (troubleshooting is impossible), leaking (internal address/topology)
- Think about the attack surface, with either you as a target (at which *rate* can people DoS you with ping? Or slow you down with source quench?) or as an accomplice (which traffic can be used for an amplification attack?)

2.7 Allow traffic from FW to anywhere, but only allow SSH access from the LAN (F)

- Remember you can never trust somebody else to do his job correctly. It's your job to secure the network, his job to secure the system. Neither should fail….
- If somebody has forget to specify to which interface to specify SSH access on an important infrastructural element, chances are that he also forgot a temporary admin:admin account :)

Please save the state of your firewall and submit it (anonymously) in the freeform answer at the end of the Google feedback form

3. Functional tests and performance benchmark

Goal: Test that your configuration works as expected, i.e., reachability of expected service, proper shielding of everything else. Don't just be nice, try also to harm your network (others will be surely less nice than you in trying doing harm)

3.1 Functional test

In the first part of your functional tests you should setup services and test if they are accessible. A comment is worth making: while you could setup real HTTP servers (simpler than the HTTPS nginx setup of Lab01)

```
root@dmz:/root# python -mSimpleHTTPServer
```

and use real HTTP clients (simpler than Chrome of Lab01)

```
root@cli:/root# wget http://dmz
```

however you should realize that your setup has no application-level gateway: i.e., you did not configure an HTTP proxy, so the use of real HTTP commands is not necessary

So you can simply listen to the port 80 with netcat (`nc`)

```
root@dmz:/root # nc -l -p 80
```

Open a TCP connection to the same port that an HTTP connection would, and write some payload.

```
root@cli:/root # nc dmz 80
GET /
```

The server will echo the payload (once you press enter to send the text) and you can also write in the listening console and have your response (200 OK) echoed back to the sender (TCP connections are bidirectional).

Test traffic for each of the A-F rules before, and check what happens in the `watch iptables -n -v -L` screen

3.2 Performance benchmark

This is not the main part of the exercice. It will not be developed much, if not to understand potential surface attacks. For instance, SNAT lets external clients access internal DMZ servers. It also hides the internal topology, which is good. However, it is open to abuse since an external

attacker can simply open lots of legitimate connections to the server, which will soon or late fill the NAT table and render the service inaccessible for new clients. You can measure how long does it take by issuing a very simple command:

```
root@rwr:/root # while true; do date; echo DoS | nc dmz 80& done
```

Alternatively, if the HTTP server is not under attack, it nevertheless share the bottleneck bandwidth with the iperf server.

```
root@rwr:/root # iperf -t 1000 -i 1 -c dmz
```

- Limiting the rate at which a given client can open NEW connections to a server can help preventing DoS (well, slowing down: the tradeoff is that it can also slow down legitimate users if the rate limiting is very low)
- Limiting the rate at which the aggregate of all possible clients can open NEW connections to a server can help slowing-down DDoS
- Limiting the transfer rate of individual ESTABLISHED connections can avoid starvation attacks

4. Refine the firewall configuration

Goal: understand how to harden the firewall configuration based on your functional and performance tests (i.e., add rate limiting). Let the configuration evolve with the addition of new services (e.g., OpenVPN, port-knocking) and be always on your guard (e.g., logging, social engineering).

Since this is the first edition of the Lab, it is possible that the first 3 parts have already taken most of the time dedicated to the Lab. Please save the state of your firewall and submit it (anonymously) in the freeform answer at the end of the Google feedback form

Given that these refinement are more difficult than the previous parts, this part is not fully developed. If instead you finished quickly and have time, pick one of these items and develop it.

4.1 Setup rate limiting

The CSO wants you to set reasonable rate limiting for transfer rate of individual connections, for arrival rate of per-host and aggregated arrival rate of new TCP connections, and for ICMP datagram as well. Discover how to do this and check your limits are enforced.

4.2 OpenVPN

The CSO wants you to deploy an OpenVPN server. Do the necessary steps to let the firewall correctly handle tun0 traffic. Since after Lab01 you know how to configure an OpenVPN server that is secure from the cryptographic standpoint, and since OpenVPN/OpenSSL version is old on Netkit, you can here use a very badly broken default Blowfish + static Diffie Hellman configuration for the sake of simplicity.

4.3 Port knocking

The CSO ask you to implement port knocking. You disagree but you go ahead and do it. If you do not know what port knocking is, you can start from this online documentation http://lmgtfy.com/?q=port+knocking

4.4 Importance of logging

(Unfortunately not feasible with the current setup). Enable logging of dropped connection attempts (or dropped packets) at your firewall. Then attack somebody's else firewall… which means that somebody is attacking *yours* as well. Guess who and how from the logging. Also if *you* know *your neighbor* port knocking sequence (social engineering), chances are that somebody else already broke into your system :)

</ RES212 Lab #2

Netfilter/iptables Firewall >