Lab 11
Firewalls

The goal of the lab is to:
 (a) Implement simple packet filters in a Linux kernel module
 (b) Realize stateless and stateful firewall rules in a simple IP network

1. Environment

Setup. You need to use two setups to perform this lab. In Task 1, you need to provision two VMs: firewall and user machines. The firewall VM will run your Linux kernel module, and the user machine will send some traffic. In Task 2 and Task 3, you will use IPMininet to emulate a simple network. To use IPMininet, you can use its Vagrant VM as we did in Lab 07.

2. Tasks

Task 1: Implementing a Firewall using Netfilter (30%)
The goal of this task is to install two packet filtering rules by implementing netfilter hooks in a loadable kernel module (LKM).

Setup
In this task, you need to work with two VMs existing on the same LAN. One of the VMs is the "firewall VM", which runs your kernel module. The other VM is used to send traffic to the firewall VM. Developing kernel modules could potentially crash your system. As a precaution, make sure that the firewall VM has no important data, and/or that you already have a backup for its data. In addition, if the firewall VM crashes, you may need to spawn a new one, so keep a copy of your code outside this VM.

Working with the Startup Code
The provided startup code implements a packet filter to block outgoing telnet traffic. Inspect the code and make sure you understand how it works. You can use the provided Makefile targets to build, clean, install and remove the module:

```
$ sudo -E make           # build the module
$ sudo -E make install_lkm # install the module
$ sudo -E make remove_km   # remove the module
```

You can check whether the module is installed by:

```
$ sudo lsmod | grep labfilter
```

Note: The provided code is tested using Linux 4.8.0-36-generic.

Testing the Startup Code

Before installing the module, make sure that you can start a telnet session from the firewall VM to the other VM (and show that in your report). Then, install the provided module and start a telnet session from the firewall VM to the other VM. Discuss your observations using proper screenshots from telnet client and Wireshark.

Implementing Other Packet Filters

Using the same code, add two hooks to block (1) the firewall VM from pinging 8.8.8.8, and (2) other machines from pinging the firewall VM. You need to show and discuss your observations after and before installing the filters, and show proper screenshots from ping and Wireshark.

Notes

(1) You can use existing Linux functions to parse packets. For example, to parse IP packets, you can write the following:

```
struct iphdr *iph;
…
iph = ip_hdr(skb);
```

If the packet does not contain an IP header, the iph variable will be NULL. Similarly, the Linux kernel provides functions to parse other headers.

(2) You can convert an IP address to a 32-bit integer for comparison:

```
struct iphdr *iph;
char match_ip[16] = "Z.Z.Z.Z"; // an IP address to match on
u32  ip_addr;     // to hold the converted IP address
…
iph = ip_hdr(skb);
// Convert
in4_pton(match_ip, -1, (u8 *)&ip_addr, '\0', NULL);
// Use ip_addr
if(iph->saddr == ip_addr) ...
```

(3) You may need to check the IP, TCP, and ICMP header files in the Linux kernel to know their definitions. For instance, the include/uapi/linux/icmp.h file contains the definitions for icmphdr structure. You can view this file online from the Linux repo:
https://github.com/torvalds/linux/blob/master/include/uapi/linux/icmp.h

(4) Finally, you can find more documentation about Linux socket buffer (sk_buff) and netfilter here:
https://linux-kernel-labs.github.io/refs/heads/master/labs/networking.html#the-struct-sk-buff-structure

Task 2: Implementing Stateless Firewall (40%)

In Task 2 and Task 3, you will use IPMininet to create a simple network and implement firewall rules using iptables in one of the routers. Recall that IPMininet enables creating fully functional IP networks inside a single machine. You should get familiar with its documentation to complete this lab.

You need to submit an individual python file, called `network.py`, that implements all requirements from Task 2 and Task 3. In your report, you need to show and discuss your observations, and include proper screenshots from mininet command prompt and Wireshark.

Setup

We provide a startup code to build the considered IP network (shown below). The network has two routers: r1 and r2 which run OSPF to calculate the routing tables. The network also has five hosts: h11–h14 and h2, each of which runs two HTTP servers on ports 8000 and 23. The goal is to install iptables rules at r1 to satisfy given requirements.
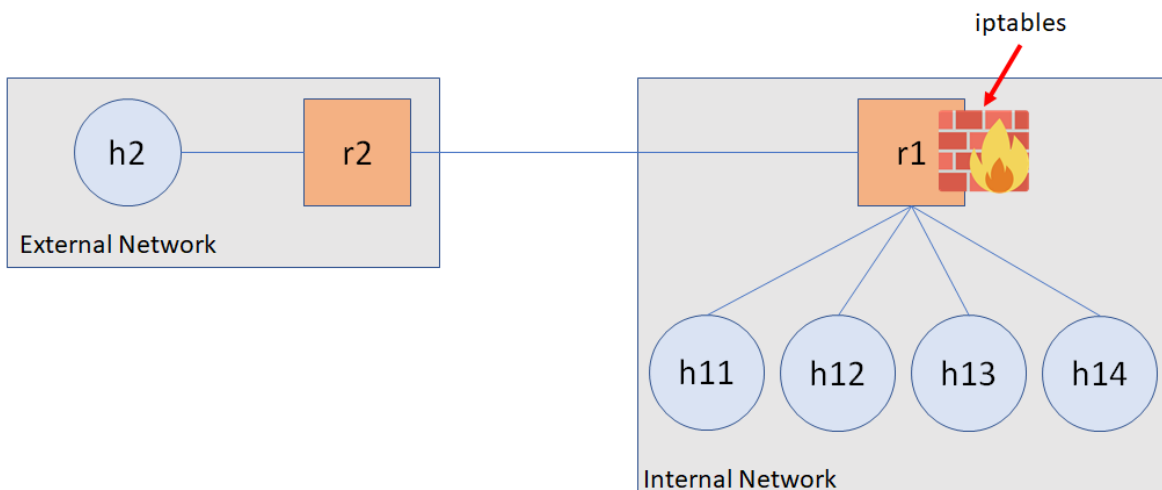

The IP network used in Task 2 and Task 3.

Implementation Note

We refer to the interfaces between r1 and h11–h14 as "Internal Network" to indicate that they belong to a single protected network. Our implementation, however, does not reflect a single network as each interface between r1 and h11–h14 belongs to a different subnet. A potential implementation is to deploy a switch "s1" to connect the four hosts as shown below. The main reason for not using the "switch" approach is that a switch in IPMininet is simply a Linux bridge. This means that iptables may not be able to perform filtering at the transport layer inside IPMininet switches (i.e., Linux bridges). Unlike an IPMininet switch, a router in IPMininet is a shell process in a network namespace with its own network stack, devices, interfaces and firewall rules. In addition, that process is configured to act as a real router by setting the IP forwarding flag.

Thus, when we say, for example, that "*h2 should not ping any host in the Internal Network*", you need to install the required rules at r1 to handle traffic on the five interfaces connected to r1.
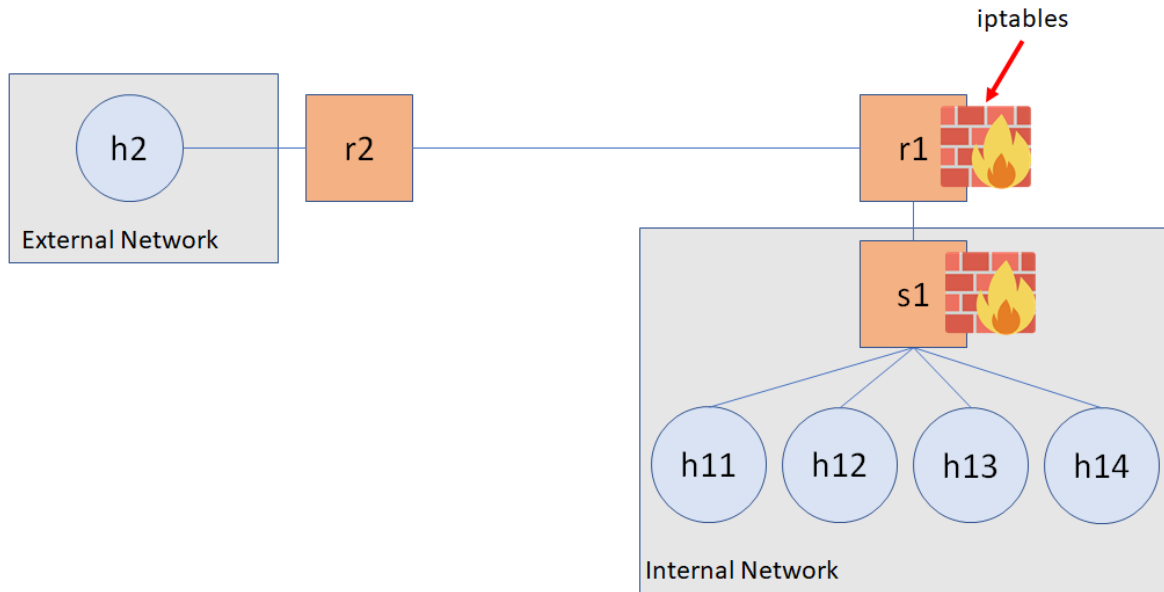
A switch-based approach to build the given network. We do not use this topology in this lab.

Your **task** is to implement stateless firewall rules to satisfy the following requirements.

Subtask 1: Protecting the Internal Network

In the first subtask, your goal is to protect the hosts in the internal network as follows:

1. External hosts (e.g., h2) cannot ping internal hosts
2. External hosts can ping r1
3. Internal hosts can ping external hosts
4. Other packets between the internal and external networks are blocked

Notice that the hosts belonging to the internal network can ping (or access) each other.

Subtask 2: Protecting the Internal Services

In addition to the requirements in Subtask 1, you need to protect the internal HTTP services by applying the following rules:

1. External hosts can only access the HTTP server on 10.11.0.2:23 (i.e., h11 on port 23), not the other internal hosts
2. External hosts cannot access other internal servers
3. Internal hosts can access all the internal servers
4. Internal hosts cannot access external servers

Note: *You should not use the conntrack module for Task 2.*

Task 3: Implementing Stateful Firewalls (30%)

As discussed in the lecture, stateless firewalls process packets independently, which may lead to undesired behaviours such as dropping packets belonging to a valid connection.

Your **task** is to implement stateful rules (using conntrack) to allow internal hosts to access the external service in h2 at port 8000.

3. Guidelines

You need to at least consider the following while writing iptables rules:
- The tables and chains to be used by iptables
- The rule order within each chain
- The traffic direction (i.e., inbound or outbound). In iptables, you need to use the "`-i <intf_name>`" or "`-o <intf_name>`" options to specify the incoming and outgoing interfaces, respectively.
- The protocol headers to match on. You can use the protocol-specific help message to print the protocol match options. For example, running the command "`sudo iptables -p icmp -h`" will show the available match options for ICMP.
- In stateful firewalls, you need to think about the right connection state to match on: NEW, ESTABLISHED and RELATED.

In addition, you may find these commands and tips useful:
- You may need to run pingall command in the mininet command prompt 1–2 times after starting the IPMininet topology. This ensures that the reachability information was calculated by OSPF.
- You need to know the interfaces' names from the mininet command prompt:

```
mininet> intfs
```
- You can interact with the iptables at r1 using the mininet command prompt. As an example, you can run:

```
mininet> r1 iptables -L # list rules
mininet> r1 iptables -F # flush rules
```
- To ping h11 from h2, you can run the following:

```
mininet> h2 ping h11
```
- You can use wget as a simple HTTP client as follows:

```
mininet> h2 wget -O - 10.11.0.2:8000
```

- Using connection tracking requires that the conntrack-tools be installed at your vagrant VM.

```
$ sudo apt-get install conntrack
$ # enable the modules
$ modprobe nf_conntrack
$ modprobe nf_conntrack_ipv4
$ modprobe nf_conntrack_ipv6
$ # check conntrack
$ sudo conntrack -L
$ sudo conntrack -S
$ # check conntrack at r1
mininet> r1 conntrack -L
```

4. Installing Rules using IPMininet

IPMininet provides a simple API to install iptables rules to a specific router. This is achieved by using the addDaemon function with IPTables configuration. For example, to install rules on r1:

```
r1_rules = [Rule(''), Rule('')]
r1.addDaemon(IPTables, rules=r1_rules)
```

The first argument to the Rule constructor is the actual rule to be installed (without the iptables string). For instance, to drop outgoing packets, you can write the following rule:

```
r1_rules = [Rule('-A OUTPUT -j DROP')]
r1.addDaemon(IPTables, rules=r1_rules)
```

5. Submission

You are required to submit:

(1) All source code you implemented to complete the tasks.

(2) A detailed report.

The files should be compressed in a single (.zip) archive.