

Firewall Lab

Copyright © 2006 Wenliang Du, Syracuse University.

The development of this document is funded by the National Science Foundation's Course, Curriculum, and Laboratory Improvement (CCLI) program under Award No. 0618680 and 0231122. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. A copy of the license can be found at <http://www.gnu.org/licenses/fdl.html>.

1 Overview

The learning objective of this lab is for students to learn how firewall works by implementing a simple personal firewall for `Minix`. A personal firewall controls network traffic to and from a computer, permitting or denying communications based on a security policy.

Firewalls have several types; in this lab, we focus on a very simple type, the *packet filter*. Packet filters act by inspecting the packets; if a packet matches the packet filter's set of rules, the packet filter will drop the packet either silently or send an "error responses" to the source. Packet filters are usually *stateless*; they filter each packet based only on the information contained in that packet, without paying attention to whether a packet is part of an existing stream of traffic. Packet filters often use a combination of the packet's source and destination address, its protocol, and, for TCP and UDP traffic, the port number.

2 Lab Tasks

In this lab, students need to implement a packet filter for `Minix`. We will call it `minifirewall`. This firewall consists of two components: policy configuration and packet filtering.

2.1 Firewall Policies

The policy configuration module is intended for allowing system administrators to set up the firewall policies. There are many types of policies that can be supported by personal firewalls, starting from very simple to fairly complex. For our lab, we will consider the following policies:

- **Block(or unblock) incoming and outgoing packets.** This policy blocks incoming or outgoing connections based on some criteria such as:
 1. *Protocol*: It specifies which protocol a policy applies to. The protocol can be TCP, UDP, ICMP or ALL. For ease of implementation, protocols can also be represented by numbers.
 2. *Source and Destination address*: Match packets with source and destination addresses. The source address is for incoming packets and the destination address is for outgoing packets. They can also be hostnames. As used by many packet filters, address/netmask combination is often used to block an address range.
 3. *Port number*: Match packets with port numbers.
 4. *Action*: Specify the actions when a packet matches with a rule. Common actions include
 - BLOCK: block packets.
 - UNBLOCK: used in conjunction with BLOCK to allow packets from just one address through while the entire network is blocked (see our examples).

- MANIPULATE: perform manipulations on packets, such as changing port numbers (see below).
- FORWARD: direct network data to a file for logging purposes.
- **Manipulate incoming and outgoing packets** Oftentimes, it is required to perform some kind of manipulations on network packets. For example, the administrator might have set the SSH server to listen on port 1403 instead of 22. In such a case, packets which are meant for port 22 have to be directed to port 1403. The manipulate option of the firewall provides this facility. Other mandatory parameter that can be manipulated is TTL (Time to live). You are free to provide other manipulation options stating their use. For usage, look at the examples.
- **Logging.** One of the hidden features of a packet filter is logging. This feature allows network administrators to monitor packet flow by FORWARD filtering data to a log file.
- **Inspection.** System administrators should have some means of finding out the policies that are currently active.

2.2 Packet Filtering

The main part of firewall is the filtering part, which enforces firewall policies. You can add the filtering functionality to Minix's network code (`inet`). You can refer to several helpful documents available on `inet` (links are provided on the lab description page).

We suggest that you first work on this packet filtering module, rather than the policy module. To start with a policy module, you can conduct filtering based on a hardcoded firewall policy. Once your packet filtering starts working properly, you can work on the policy implementation and integrating policy with filtering.

3 Example Usage

This section shows some example usage of our firewall. Feel free to change the syntax according to your own convenience.

- `minifirewall -PROTO ALL BLOCK`
Block all packets.
- `minifirewall -PROTO TCP UNBLOCK`
Allow only TCP data.
- `minifirewall -ADDR 172.16.75.43 -PROTO ALL -A INCOMING BLOCK`
Block all incoming packets from the given IP address.
- `minifirewall -ADDR 172.16.75.43 -NETMASK 255.255.0.0 -A INCOMING -PORT 80 -PROTO TCP BLOCK`
Block all incoming TCP packets from addresses `172.16.*.*` that are directed towards port 80.
- `minifirewall -PROTO ALL MANIPULATE -ORIGPORT 22 -NEWPORT 1403`
Redirect all packets meant for port 22 to port number 1403 as the SSH server is configured to listen on that port.

- `minifirewall -PORT 80 -PROTO ALL -LOGFILE HTTPLOG FORWARD`
Log all traffic to and from port 80 on host machine to a file called HTTPLOG in the current directory.
- `minifirewall -A ALL PRINT`
Print to screen all active rules.
- `minifirewall -Z`
Flush out all rules.

4 Suggestions

We have compiled a list of suggestions in the following. Please read them carefully before you start the labs.

1. *An important distinction.* Before you start coding your firewall, it is essential to focus on design. A proper approach to designing is to make a distinction between mechanism and policy. While mechanism provides the different ways an action can be performed, policies defines the actions to be performed. With reference to this lab, packet filtering is a mechanism. whereas filtering rules are policies.

To better explain this important distinction, consider that we select a design where packets that are sent to and from `127.0.0.1` are always ignored in the `inet` code. This is a not-so-good design because we are imposing a restriction on the mechanism by putting a block on the kind of packets that can be filtered. Instead, a better approach is to let the user decide what to do when a packet is from and to `127.0.0.1`.

2. *Code Reading.* To read Minix source, it is quite inconvenient to do so in the Minix environment because of the lack of tool support in Minix. We suggest that you copy the entire source code to your host machine (Windows or Linux), and use code-reading tools that are available on those platforms. All the source code of Minix can be found under the `/usr` directory. We also put a copy of the entire source code on the web page of this lab.

Browsing source code of Minix is not easy, because source code is in a number of directories. Sometimes, it is quite difficult to find where a function or data structure is defined. Without right tools, you can always use the generic search tools, such as `find` and `grep`. However, many of our past students have suggested a very useful tool called *Source Insight*, which makes it much easier to navigate source code of a complicated system. It provides an easy way to trace function and data structure definitions, as well as other useful features. This software can be found at <http://www.sourceinsight.com>. Another choice for browsing source code is to use the online Minix source code at <http://chiota.tamacom.com/tour/kernel/minix/>.

3. *How Minix Networking Works (I).* Understanding how networking works in Minix is essential for this project. Several helpful documentations are available. In particular, we highly recommend the documentation at <http://www.os-forum.com/minix/net/>, which provides a line-by-line analysis of Philip Homburg's network service for Minix, version 2.0.4 (the version that we use in this lab is version 3, which is not so different from the version 2.0.4 in the networking part). Our past students found the documentation very useful. Please focus on two files: `ip_read.c` and `ip_write.c`. All outgoing IP packets are processed in `ip_write.c`, and all incoming IP packets sent to up layers (TCP/UDP) are processed in `ip_read.c`.

4. *How Minix Networking Works (II)*. We have developed a document to further help you understand how the Minix networking works. The document can be found at the lab web site. It guides you through several source code to show you a big picture on how a packet is forwarded from application to ICMP/TCP/UDP to IP, and then to Ethernet. It also describes how `add_route.c` and `pr_routes.c` works. These last two files (in `/usr/src/commands/simple`) can serve as a good example on how to store and maintain (routing) information in the kernel. If your need to do the similar thing (i.e., storing information in the kernel), you can use the system calls in `inet`, such as `ioctl()` in `ip_ioctl.c`, which need to be changed to add more functionalities. The files `pr_routes.c` and `add_routes.c` give you a good example on how to use the system calls.
5. *Testing*. Testing is an important step of this lab to make sure that your firewall performs according to expectations. There are two main aspects to testing:
 - (a) Testing whether policies give desired results: For each of the policies that you have implemented, make a list of commands that utilizes these policies. Run each of the commands in your list and check if they produce desired results. Some tools that will help you in this process are `Wireshark` (<http://wireshark.org>) and `Fttester` (<http://dev.inversepath.com/trac/fttester>).
 - (b) Checking for system stability: You should make sure that your firewall does not make your system unstable or cause a system crash. You should always be very careful about freeing unused memory. Run your firewall long enough and feed it a wide variety of rules so that you are sure that it does not kill your system!

5 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstration:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.