The Complete Documentation of My Firewall – Ex3

In this assignment, I had to implement a stateless firewall that filter packet based on a static rule table.

The firewall consist of 2 main parts: kernel mode and user mode.

There are 4 mechanisms the FW support:

* Loading and Storing rules the rule table.
* Sending to the user space the rules.
* Showing the filter log to the user space.
* Reset the log.

Kernel Mode

For the FW, we use netfilter to hook packets and apply the filter, and 2 Sysfs devices for interactions with the user mode.

The kernel module consists of 4 files:

* Hw3secws – Here we initialize the the char devices, sysfs classes and the netfilter hook point.
* FWLogDevice – Here we have all the functions and mechanisms related to log actions used in the log devices.
* FWRuleDevice – Here we have all the functions and mechanisms related to the rule table we used the rule device for.
* PacketHandler - Here we have the hook point filtering function and filtering related functions.

hw3secws

when the module load, it start by running the *my\_module\_init\_function* which do the following:

1. Creating a sysfs class we will attach to the log and rule device.
2. Creating a log char device and also a sysfs device we add to the recently created class.

For the log char device, we initialize it with our implementation of open and read we will use later for sending the log table to the user mode.

For the log sysfs device, we initialize it with a store function only, we will use to get a signal from the userspace for reseting the log table.

1. Creating a rule sysfs class device and add it to the sysfs class.

We will use the store function to get the rules from the user and show function to send them to the user space.

1. Initializing a forward hook point with a filtering function we will describe later in the filter file.

FWRuleDevice

In this file, we manage the rule table.

This file contains 2 main functions, the first is store\_rules and the other is show\_rules.

The function show\_rules is responsible for sending the rules in the rule table to the user. It uses auxiliary function to convert every rule to compressed buffer and send it to the user.

The function store\_rules, similar to show\_rules, is responsible for receiving the rules from the user and store it in a static rule table, we use in Handle\_packet for filtering packets.

There are some other "side" functions implemented in the file but they are very simple and documented in the code itself.

FWLogDevice

In this file, we manage the logging system. I used the struct list, that already implemented in the kernel for logging. It allows me to "travel" through logs and more importantly, its dynamic, so we are not limited to constant size.

This file contains 2 main functions, the first is the add\_log function, and the other is read\_log.

Add\_log is a function that given a log and reason, it adds the log to the "log list", where we store all our log". In a case where a similar log is already exist, it increases the already existed log counter by 1. I used some function that implemented already in the kernel for the struct list. More information about the struct list can be found here: https://medium.com/@414apache/kernel-data-structures-linkedlist-b13e4f8de4bf

Read\_log is my implementation for the read function in the log device we create in the hw3secws module. This function is called when someone, the user in our case, trying to read from the log device. The function "traveling" through the link list that contains the logs and convert each log to compressed buffer and send it to the user.

There are some other "side" functions implemented in the file but they are very simple and documented in the code itself.

PacketHandler

In this file, there is the implementation of the Handle\_Packet function, which is the function we use in the hook point to filter packets.

The function Handle\_Packet first get the desire information from the packet and then first check if the packet is loopback or the protocol is not TCP/UDP/ICMP so we need to accept the packet without logging. It check for xmas packet and if the rule table is not valid. In this case it dropes the packet and log it with the right log reason. After we cover all the "special" cases, we iterate over the rules and check if there is a match. If there is a match, follow his action and else we drop the packet.

There are some other "side" functions implemented in the file but they are very simple and documented in the code itself.

User Mode

In the user mode, we has a program that interacts with the FW Kernel module and get/send the right information according to the user commands.

The user module consists of 3 files:

* UserInterface – This file contains only the main function of the user program, which check parse the input and run the corresponding function.
* UserLogManager – Here we have all the functions and mechanisms related to log actions used in the user mode, like printing the logs and receiving them from the kernel.
* UserRuleManager – Here we have all the functions and mechanisms related to rules actions used in the user mode, like receiving/sending the rules to the user/kernel.

UserInterface

In this file, we have the main function, that parse the input from the user and run the functions according to the user intput. We have support for 4 commands:

* Load\_Rules <Path> – This command will send to the kernel the rule table in Path.
* Show\_Rules – This command will get the current rules table from the kernel and print it to the user. In case of no table is already in the kernel, we print a message that there are no rule table.
* Show\_Log – This command will get the current logs from the kernel and print it to the user.
* Clear\_log - This command will clear the logs in the kernel and empty it. Showing the log right after clearing it will print nothing.

UserRuleManager

This file has implementations for the Load\_Rules and Show\_Rules commands.

The Load\_Rules function, which is called when the user insert the command Load\_Rules, is reading all the rules in the provided path, and converting it the rule structs. Then it converts all the rules to compressed buffers and send it to the rule sysfs class in the module.

The Show\_Rules function which is called when the user insert the command Show\_Rules, is trying to read from the rule sysfs device the amount of rules, then if there no rules (the amount is 0), we print to the user that there are no rules. If there are, we read all the rules (which will be sent as a compressed buffer) and then convert it to rule structs and to strings from there. At the end it prints all the rules to the user.

UserLogManager

This file has implementations for the Clear­\_log and Show\_log commands.

The Show\_Log function is called when the user insert the command Show\_Log. we read from the log char device (and not the sysfs like the others, since the logs can be way bigger than a page size) all the logs, which will be sent as a compressed buffer, and then convert it to log structs and to strings from there. At the end it prints all the logs to the user. In case of no logs (like after clearing the logs), it will print empty log table.

The Clear\_log function is called when the user insert the command Clear\_log. This function write to the log sysfs device. The device on writing will run the reset\_log function that clears the log.