Workshop in Information Security – Homework #3

Goal:

- Create **real useful** Firewall
- Implement Stateless Packet Filtering in the Firewall
- Be familiar with network packets and their representation in Linux kernel
- Understand the enforcement of Firewall rules in the real world
- Create the infrastructure for the next levels

Description:

In this assignment, we will start to implement our Firewall (you can ignore all the work from previous assignments, keep only the lab environment). We will start with Stateless Packet Filtering.

As we learn, Stateless Packet Filtering is a way to protect the network by enforcement of a static rule table against each packet arriving to the Firewall, individually. Each packet arriving to the Firewall should be compare against the rule table and look for a match. If there is a match, the Firewall should act as the rule says, accept or drop.

Every rule in the table should be like we learned and appears in the attached fw.h file.

You should locate your hook function only on “forward” point. From this step we will ignore other hook points like local in or out.

It’s highly recommended to go over fw.h file before starting the implementation. You can change or add stuff to this file, except changing `log_row_t` and `rule_t` structures. Pay attention on the comments!

The logic:

Inspection logic: each packet arriving the Firewall should be looking for a match in the rule table and act according to the specified action of the matched rule. Log the action with the packet’s data. Aggregate the logs: if similar packet (meaning same IP, ports, etc.) already matched and has a log, only increase the counter on the existing log and do not create a new entry. The timestamp should be the last time this entry changed.

Loading rule table to the kernel: pass the data to the kernel, override previous table if exist. Locate the data in the right place in the rule table.

Showing rule table in user space: pass the data to the user space. Try to pass as less data as you can over the drivers and make the output to user readable in the user space program.

Showing logs in user space: same as above.

Resetting log table: pass any data to the specified interface, make sure to cleanup everything that not needed anymore.

The drivers:

Description	Driver path	Permissions	Passing data
Loading and showing rules table	`/sys/class/fw/rules/rules`	RW	List of rules
Showing logs	`/dev/fw_log`	R	List of logs
Resetting logs	`/sys/class/fw/log/reset`	W	N/A

Data structures:

Basically, you can use whatever data structure you want, but of course it should be correct. For example, you can't use static table for the logs, because it may be very large table and should be allocate by demand. Below are my recommendations.

For the rule table, use static table. The rule table is limited to MAX_RULES which defined 50. So you can create a table with size MAX_RULES, just make sure to remember how many rules are in use.

For the logs, use "klist" which is linked list implementation for kernel. You can also use dynamic allocation by demand. Avoid allocation for a single entry!

User space program:

Write a user space program that will communicate with the kernel module. The program should get an arguments and work by them. The supported values are:

- show_rules
- load_rules <path_to_rules_file>
- show_log
- clear_log

The name of the executable will be "main".

Christmas tree packet:

A TCP packet with FIN URG PSH flags on (all of them) called Christmas tree packet. This kind of packet is not legitimate TCP packet and **you should drop it** with reason `reason_t.REASON_XMAS_PACKET`. Refer the internet to learn more about Christmas tree packets.

Important notes:

- Do not trust user! Validate the inputs within the kernel module (except of course the necessary validation in user space and dumping error messages on invalid input)
- Implement as modular as possible
- Minimize parsing work in the kernel (relevant for both incoming data to kernel and outgoing from kernel)
- The name of the module should be firewall.ko
- If no rule matched, **drop the packet**
- Store the data in a way that minimize the casting between LSB and MSB in runtime
- The eth1 interface (connected to host1) is the **IN** and eth2 interface (connected to host2) is the **OUT**. Use this information for the "direction" field.
- Accept any non TCP, UDP and ICMP protocol without logging it. Same for IPv6 packets
- Accept any loopback (127.0.0.1/8) packet without logging it
- In the log, when "reason" is not one of the members of `reason_t` structure, it should be the rule index number
- **See examples for input rule table and output rule table and logs; it should be exactly like this!**

Submission

Prepare a ZIP file, contains:

- "module" folder, includes hw3secws.c file (the module) and the Makefile
- "user" folder, includes the user-space program and (if needed) the Makefile
- Dry documentation

General rules for submission, valid for the next assignments as well:

- Document your code
- If you use a code from the internet, document it and add the source
- Individual submission
- If needed, split the files into a modular files

Good luck!

