Operating Systems
Semester 1, 2014
Eike Ritter

The University of Birmingham
19 November, 2014

Assessed Exercise 4

Deadline: Monday 8 December, 4pm

The Task

Write a kernel module which tracks incoming TCP connections to a set of up to 64 user-defined destination ports. For each destination port, it should store up to 32 of the most recent (source addresses, source port) pairs from which connection was attempted. See below for important instructions how to test your kernel module.

The kernel module should create a proc-file `/proc/iptraffic`, which supports the following operations:

- a userspace program may write a string to the proc-file containing a comma-separated list of destination ports to be monitored, for example:

  ```
  80,110,25
  ```

  If the input is not in this format, contains more than 64 ports, or is otherwise invalid, then -EINVAL should be returned.

  When the list of ports to be monitored is written to the proc-file, it should completely replace the list currently held in the module, if any.

  For simplicity, there is no requirement to detect or prevent the simultaneous writing of two sets of rules by two writers.

- a userspace program may read from the proc-file to get the currently-held information, which should be supplied as a string in the form:

  ```
  BEGIN\n
  80:35108:82.70.211.182\n
  80:34109:82.70.211.182\n
  110:34103:82.70.211.182\n
  END\n
  ```

  If the buffer provided for the read() is not big enough to hold all the data, as much as possible should be returned.

Your code needs to handle multiple readers and multiple incoming TCP-packets at the same time.

Note that the firewall hook runs in interrupt context when called for incoming packets. Hence it must not sleep. You will therefore need to study carefully which kernel functions may sleep, and make sure that you do not use these in the firewall hook. Also, you need to ensure that you do not sleep while you hold a lock (for example kmalloc).

An example kernel module which checks incoming packets is available via the website.

Testing

Because the virtual machine which is used for the module uses network address translation, no incoming connections are possible by default. Hence you should use test your kernel module by initiating outgoing connections from your virtual machine via `telnet`. Your kernel module should then check for the acknowledgement packet sent in reply to the first outgoing packet. The destination port of this acknowledgement packet is the port on the virtual machine which is used for this connection. Hence in your module the role of destination and source port are swapped—see the example module which rejects all connections to port 80 (in other words, web-traffic. We will use outgoing connections for marking your code as well.

A test process `testip` will be used to test your submission, and it will also be provided to you to aid in your testing. It will be invoked by:

- `testip R` (to read the currently-held data using a read() call); or
- `testip W 110,25,80` (to update the list of monitored destination ports using a write() call)

Marking Scheme

Please use the supplied userspace program `testip.c` to test your work; this program will be used to assess your submission. Submit your code via Canvas, but only the source code and makefiles for the kernel module. We will compile and run your code on the virtual machine and mark it accordingly.

Please in particular note that we will use the compiler option introduced in the lecture and deduct 6 marks immediately if there is any compiler error or warning. You should submit precisely ONE file via Canvas, in TAR format, containing a top-level directory "assignment4" in which should appear all the files comprising your submission. Do NOT use a complex directory structure below this.

The end result of invoking your makefile with the "all" option should be precisely ONE kernel module called "firewallInput.ko". The output file MUST be in the top-level directory after your makefile completes; if it is not, your work cannot be tested.

We will award marks as follows:

- 4 marks for correctly handling reading and writing from/to the proc-file.
- 4 marks for correctly managing the list destination ports to be monitored.
- 6 marks for correctly processing the incoming connections and accumulating the monitoring data.
- 6 marks for correctly handling the concurrency.