Advanced Networking 2018

Lab #6: eBPF
Total points: 10 pts

Assignment
Lab date: March 16 (with spillover date: Tuesday 20) 2018
Submission date: 11:59PM March 20 2018 CET

Authors: Łukasz Makowski, Ralph Koning
Email: l.s.makowski@uva.nl, r.koning@uva.nl
University of Amsterdam

Abstract

In this assignment you will get acquainted with extended Berkeley Packet Filter (eBPF) technology. First, we will go over fundamental principles of this Linux kernel subsystem. Next, you will create your own programs performing network packet operations.

Preparation

1. Boot into your normal Ubuntu installation first (no Xen).
2. Execute the following to install the Vagrant environment and to load the virtualbox kernel modules:

   ```
   sudo apt install vagrant virtualbox wget
   sudo modprobe vboxdrv vboxnetadp vboxnetflt vboxpci
   ```

3. Download and setup the Vagrant box and assignment materials:

   ```
   wget http://amiens.studlab.os3.nl/an2018/lab-ebpf/an-ebpf.tgz
   tar xf an-ebpf.tgz
   cd an-ebpf
   vagrant up #(it will download the VM image and start it so have some patience)
   ```

4. Log-in to the VM and verify if the test eBPF code compiles:

   ```
   vagrant ssh
   sudo su
   cd /vagrant
   cd src
   make #the output command should state 'SUCCESS' in the last line
   ```

There is some literature that you might want to consult in the upcoming tasks:

- http://www.tcpdump.org/papers/bpf-usenix93.pdf, with focus on subsections 3.3-3.5;
- https://www.kernel.org/doc/Documentation/networking/filter.txt), with the details regarding BPF implementation in Linux;
- http://cilium.readthedocs.io/en/latest/bpf/#llvm, containing the Cilium project documentation;
- *tc-bpf* and *bpf syscall* manual pages briefly answer the questions below.

Task 1: Compiling and usage of sample eBPF program (2 points)

To get a grasp of how the eBPF program is used in practice we will first take a sample eBPF program. The Linux kernel contains multiple eBPF examples written in restricted C, those can

be found in *linux-4.15/samples/bpf/*. We annotated an example for you, this can be found in *src/bpf_sample.c*. Open the file and try to understand what it does.

- Q3.1 - Look at the code section SEC("rewrite_tcp") and explain the code within. Be specific, explain the role of used function parameters and the purpose of returned values (look inside *linux-4.15/tools/include/uapi/linux/bpf.h* for the documentation).
- Q3.2 - Compile the program using *tools/bpf-compile* wrapper: *tools/bpf-compile <filename>*. Apply the program to *eth1* interface using: *./tools/bpf-tc eth1 bpf_sample.o rewrite_tcp*
  Verify that the object file is loaded and provide the output of the above command.
  Hint: use *tc*.
- Q3.3 - Illustrate the functionality realised by the attached program. Use tools such as *ping* or *nc* to generate sample packets and *tcpdump* for your packet traces.

  Note: you won't be able to see your ebpf-modified packets if you use tcpdump on the same interface as eBPF object file. The best way to see these packets is to:

  1. Create a dummy net interface
     'ip link add dummy0 type dummy'
     'ip link set dev dummy0 up'
  2. Mirror your packets to dummy0 In your eBPF sample code, change the interface index from 255 to if index of the dummy interface. You can look up the if index with *ip link show*
  3. Use tcpdump to listen for your traffic:
     'tcpdump -i dummy0 -nnn -vvv'

Task 2: Writing eBPF program: traffic firewalling (3 points)

One of the use-cases for BPF programs is packet filtering. A network packet can be matched against a specific field and the decision to accept, drop or redirect it can be taken. Implement your filtering program in the sample by creating your own, it can be as simple as accepting only specific protocols (e.g. TCP+IPv4 only) or verifying port numbers and IP addresses. Use the section SEC("task4").

Use *tcpdump* to verify that it works.

Task 3: Filtering performance measurements (3 points)

In this task you will compare the filtering performance you can achieve with iptables and eBPF against a simulated DDoS attack.

Note: If all goes right, iptables should have a big performance impact while eBPF does not. You are running this in a virtual environment so your milage may vary. Therefore, even when not seeing performance difference, a correct execution of the experiment gets you all the points.

Check out the sample eBPF code filtering out the source IPv4 addresses `src/ip_filter_w_map.c`. Compile it and attach to an eth1. Using *bpf-map* tool that should be installed on the vagrant vm (https://github.com/cilium/bpf-map) verify that `ddos` map has been exposed in the operating system as `/sys/fs/bpf/tc/globals/ddos` file. Remove the eBPF object file from the interface before following with the rest of an assignment.

- Q3.1 - Execute `iptables -t raw -i eth1 -A PREROUTING -j NOTRACK` on the vagrant VM to prevent 'conntrack table' space exhaustion. Start an iperf3 server on the VM and run `iperf3 -c 192.168.2.100 -t 70 -O 10` on the host machine to test the bandwidth between the host and VM. First, run it as is to see the raw performance. Also on the host system, run `hping3 --rand-source 192.168.2.100 --faster` command to start generating DDoS traffic; leave hping3 running it the background. If hping3 kills your connection, change `--faster` to `-i u1000` and change the number to increase or decrease the sending rate of hping3. The goal is to see slight degradation of iperf speeds (caused by hping3) without any rules applied. Show what you did, and that a hping3 is properly tuned.
- Q3.2 - On the VM, use `tools/load_rules` script to load filtering rules for both iptables an eBPF program. First run it with 'ipt' argument, it creates a new (unreferenced) chain named ddos and fills it with sample IPv4 addresses stored in 10k_random_ip.txt file. Measure the performance when all the incoming traffic goes through 'ddos' chain i.e. `iptables -i eth1 -I INPUT 1 -j ddos`. Show that the chain is applied and is receiving traffic, also include the output of your measurements. When done, don't forget to remove the rule.
- Q3.3 - Attach the eBPF object file, and fill the eBPF map using the `tools/load_rules` script. This time, use the 'ebpf' argument, to fill the 'ddos' bpf-map with the hex versions of the previously loaded addresses. Show that the map is applied and is receiving traffic. Repeat the previous measurement and include the output and results. Did you notice performance difference?

Hints:

- `tools/ipv4_to_hex` can be used to convert IPv4 address dotted format to hex
- `tools/show_map_nonzero` lists the map content which has the counter value > 0

Task 4: Understanding eBPF architecture (2 points)

- Q4.1 - What is the difference between (c)BPF and eBPF?
- Q4.2 - What is the purpose of eBPF maps?
- Q4.3 - How does an eBPF program gets passed to the kernel? Which user-level tools are used for this?
- Q4.4 - What kind of operations can be performed on a network packet inside eBPF code?

Submission

Submit the following file:

- **report** (PDF format) that describes in detail steps performed and answers.

  `lab6-report-$LastName.pdf`

Any other kind of submission will not be taken into account.