1 INTRODUCTION

Nowadays, the traditional security features of a Linux system are centered on iptables, which has been the most used packet filtering mechanism in the Linux kernel for almost 20+ years. However, the increase in network speed and the transformation of the type of applications running in a Linux server has led to the consciousness that the current implementation may not be able to cope with the modern requirements particularly in terms of scalability, as the number of rules is dramatically increasing [2].

In recent years, the extended BPF (eBPF) subsystem has been added to the Linux kernel, offering the possibility to execute (almost) arbitrary code when a packet is received or sent, including stateful processing. Notably, this does not require any additional kernel module and offers the possibility to compile and inject this code dynamically, hence facilitating over-the-air updates. The above characteristics make eBPF a perfect candidate to build an iptables clone such as [1], which can be considered more an initial proof-of-concept that filters traffic based on IP addresses than a full iptables replacement. This paper starts from the above activities and presents a first eBPF-based prototype, bpf-iptables, which emulates the iptables filtering semantic and exploits a more efficient matching algorithm. Finally, we evaluate our prototype comparing it with the current implementation of iptables, showing how this allows obtaining a notable advantage in terms of performance particularly when a high number of rules is involved, without requiring custom kernels or invasive software frameworks (e.g., DPDK) that could not be allowed in some scenarios (e.g., servers in large datacenters).

2 BPF-IPTABLES DESIGN

The design of bpf-iptables includes two orthogonal aspects: (i) the strategy adopted to preserve the semantic of the iptables firewall policies, and (ii) the data plane architecture, which is driven by the algorithm used for packet matching. We leave the detailed techniques and implementations to a future paper and focus here on the key design of the bpf-iptables architecture.

Iptables filtering semantics. Iptables filters packets either in the INPUT, FORWARD and OUTPUT chains of the Linux Netfilter [5] framework, which are located in a different position compared to eBPF hooks (Figure 1). In particular, the XDP hook, available only for incoming traffic, is located at the earliest point in the networking stack. Instead, the Traffic

![Figure 1: Netfilter vs eBPF hooks](image)

Control (TC) hook, available for both incoming and outgoing traffic, is executed either before the PREROUTING or after the POSTROUTING hook. The different position of the filtering hooks in Netfilter and eBPF poses non-negligible challenges in preserving the semantic of the iptables rules, which, when enforced in an eBPF program, operate on a different set of traffic compared to the one that would cross the chain they are attached to. As an example, a rule “iptables -A INPUT -j DROP” drops all the incoming traffic directed to the current host, but it does not affect the traffic that is being forwarded by the host itself. A similar “drop all” rule, applied in the ingress XDP/TC eBPF hook will instead drop all the incoming traffic, also the one that would be forwarded by the host itself. bpf-iptables adds an XDP/TC Ingress Chain Forwarder and a TC Egress Chain Forwarder eBPF program (Figure 2) to recognize the actual path of each packet and emulate the iptables filtering behavior, redirecting the packet to the proper filtering (INPUT or FORWARD) chain.

Matching algorithm. Improving the existing linear search of iptables does not appear so difficult. However, many of the existing matching algorithms (e.g., cross-product, decision-tree approaches) require either sophisticated data structures that are not available for eBPF programs [4] or an unbounded amount of memory, which is not desirable for a kernel program. bpf-iptables uses the Linear Bit Vector Search [3], which proved to be reasonably fast while being feasible with current Linux kernels (and available eBPF maps).

The algorithm consists in creating a specific bi-dimensional table for each field on which packets may match, such as IP addresses, TCP/UDP ports, and more. Each table contains the
Figure 3: Example of the LBVS classification pipeline

3 EVALUATION

We evaluated bpf-/iptables by attaching it to both the TC ingress and egress hooks of the host interfaces and compared its performance and expressiveness toiptables in two different cases. In the first test, shown in Figure 4(a), we added an increasing number of rules to the ingress chain of the firewall and we measured the performance of both applications, and then we calculated the resulting TCP throughput. In both tests, we generated traffic so that only one CPU core is involved in the processing. Results confirm that bpf-/iptables outperforms iptables in all cases.

Figure 4: bpf-/iptables performance comparison

(a) UDP throughput (forward)
(b) TCP throughput (input)

REFERENCES