Accelerating Linux Security with eBPF iptables
Matteo Bertrone, Sebastiano Miano, Fulvio Risso, Massimo Tumolo
Department of Control and Computer Engineering, Politecnico di Torino, Italy

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 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
list of unique values for that field present in the given ruleset,
plus a wildcard. Each value in the table is associated to a
bitvector of length \( N \), equal to the number of rules, which
keeps the list of rules matching that field. Finally, a bitwise
AND of the intermediate bitvectors returns the list of matched
rules; the one with the highest priority (corresponding to the
first rule with 1 in the bitvector) is returned. Each map can
be implemented in a different way, based on the field charac-
teristics (e.g., longest prefix match in case of IP addresses
and ranges; hash tables for TCP/UDP ports). Per-CPU maps
are used whenever possible to avoid cache pollution among
different CPU cores and increase the effectiveness of paral-
lel processing of multiple packets on different CPU cores.
Thanks to the dynamic code injection of eBPF, we created a
matching pipeline that contains the minimum number of pro-
cessing blocks required to handle exactly the fields required
by the current ruleset, avoiding unnecessary processing for
unused fields. New processing blocks can be added at run-
time if the matching against a new field is required, always
keeping the optimal number of programs.

**Data plane architecture.** Figure 2 shows the overall data
plane architecture of bpf-iptables. When a packet reaches
a given port, it triggers the Ingress Chain Forwarder program
attached to the TC or XDP hook; depending on whether
the packet is directed to a local application, it jumps to ei-
ther the INGRESS or FORWARD chain, which implement the
corresponding matching pipeline. Before entering the respec-
tive chains, the packet passes from a first connection tracking
module (implemented as an additional eBPF program), which
associates a state to each packet, so that all subsequent chain
rules can be correctly applied. When the packet leaves the
chains, as result of an ALLOW action, it passes through a sec-
ond conntrack module that saves the state of the session in
its internal eBPF map; depending on the chain, the packet is
then delivered to the Linux stack or forwarded to the output
port. On the other side, when a local application sends a
packet out to a netdevice, it will trigger the Egress Chain
Forwarder program attached to the TC egress hook that,
looking at the source IP of the packet decides to forward it
directly to the output port (because the FORWARD chain has

![Figure 3: Example of the LBVS classification pipeline](image-url)

![Figure 4: bpf-/iptables performance comparison](image-url)

Figure 4: bpf-/iptables performance comparison

already processed it) or to jump to the OUTPUT chain where
the classification algorithm will be applied.

## 3 EVALUATION

We evaluated bpf-iptables by attaching it to both the TC
ingress and egress hook of the host interfaces and compared
its performance against iptables in two different cases. In
the first test, shown in Figure 4(a), we added an increasing
number of rules to the FORWARD chain of the firewall and we
generated a unidirectional stream of 64B UDP packets. In
the second, shown in Figure 4(b), we added an increasing
number of rules to the INGRESS chain to protect locally run-
ning 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 by an order of
magnitude when a high number of rules is used, thanks to
its improved algorithm and the different optimizations on
the classification pipeline that are allowed by the dynamic
code injection of eBPF, with a vanilla Linux kernel.

## REFERENCES

747504/

https://cilium.io/blog/2018/04/17/
why-is-the-kernel-community-replacing-iptables

forwarding using efficient multi-dimensional range matching. In ACM

Creating Complex Network Service with eBPF: Experience and Lessons
Learned. In High Performance Switching and Routing (HPSR). IEEE.