BPF to eBPF, the new Linux Superpowers

What is BPF?

Tcpdump is a magical command that often comes to rescue most network engineers by revealing what's going on the wire. The observability becomes especially important when things go wrong, even for the most basic debugging. Tcpdump came to life with seminal work of Steve McCanne and Van Jacobson in 1992 when they published a paper describing a better way of filtering packets in the kernel and called it BPF or The BSD Packet Filter. The idea was to capture useful network packets directly from the kernel, without copying them to userspace and then funneling them out through a network tap. This allowed discarding unwanted packets as early as possible.

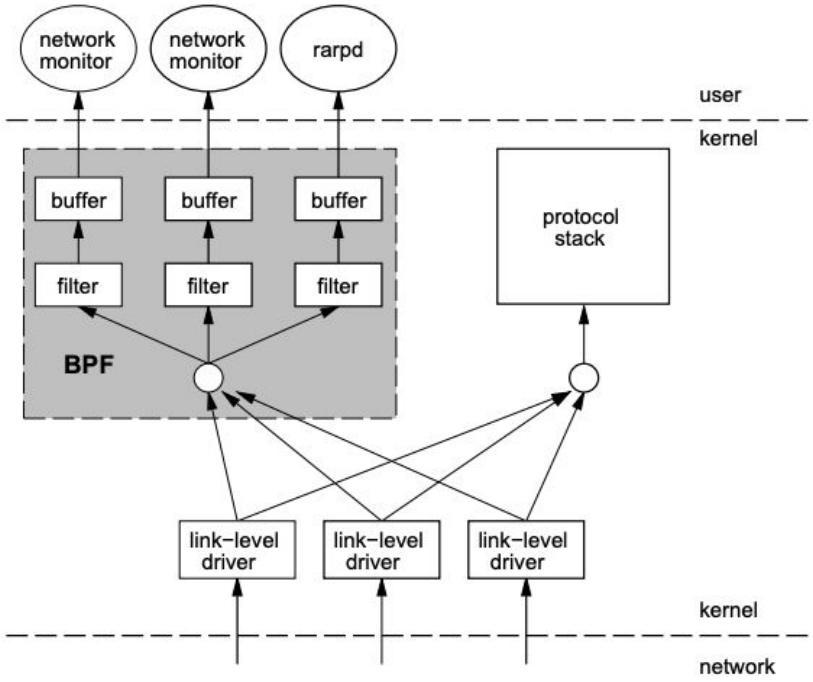


Figure 1: BPF Overview

BPF has two main components: (a) the network tap and (b) the packet filter. The network tap collects copies of packets from the network device drivers and delivers them to listening applications. The filter decides if a packet should be accepted and, if so, how much of it to copy to the listening application.

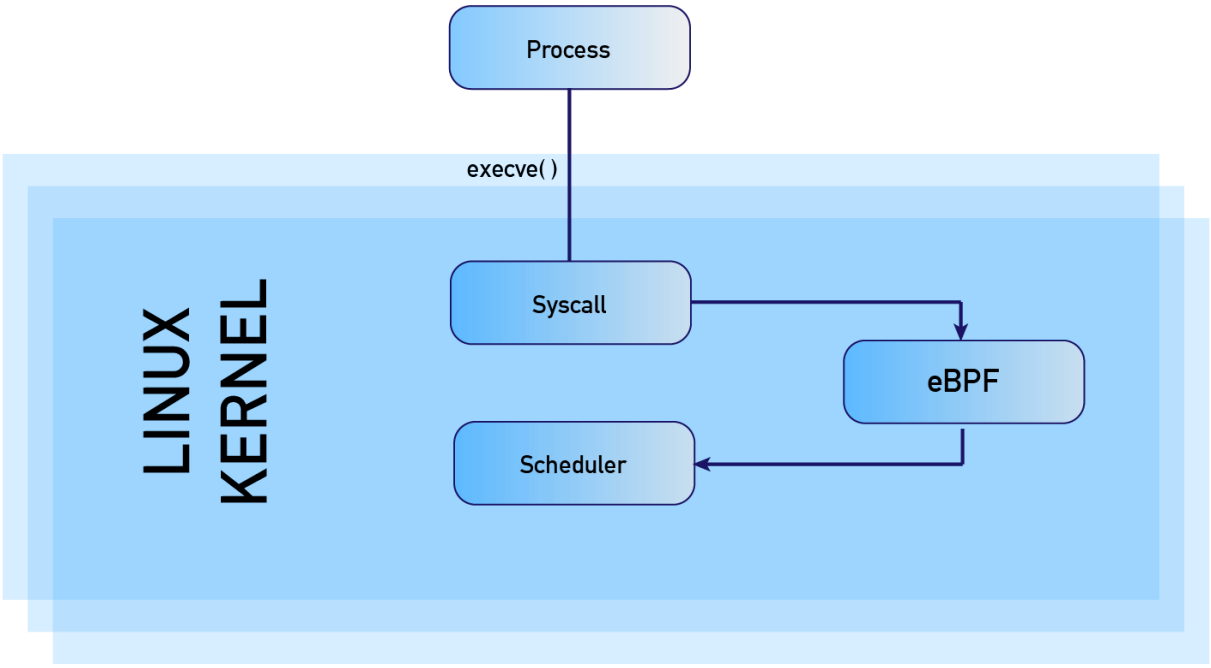
The rest is history as tcpdump, and libpcap became a defacto standard in network debugging.

What is eBPF?

After the success of BPF, the community wanted to extend the same power of BPF to other areas like monitoring, networking, and security, which led to eBPF or Extended BSD Packet Filters often called Extended Berkeley Packet Filters. Another way to compare eBPF with something contemporary is to look into Javascript. Traditionally, websites were static and mundane. Javascript enabled programmers to run mini-programs on events like a mouse click. The mini-programs run in a safe VM inside a browser.

eBPF did something similar to kernel programming. The programmers can write mini-program than run on certain events like socket connections, disk I/O, function entry/exit, kernel tracepoints, network events, and several others. These run in safe VM in the kernel and are part of the Linux kernel, ensuring that eBPF cannot crash a program or kernel.

eBPF offers pre-defined event hooks like system calls, function entry/exit, kernel tracepoints, network events, and several others.



If hooks are non-existent, it is possible to create new hooks, kernel probe (kprobe), or user probe (uprobe) to attach eBPF programs anywhere in the kernel or user application. The Linux kernel expects eBPF programs written in the form of bytecodes. Compilers are used to accomplish this task. The bytecodes are compiled into machine-specific instructions Just-in-Time (JIT) to optimize the program's execution speed. JIT makes eBPF code run at equivalent efficiency as natively compiled kernel code or code loaded as a kernel module.

Is eBPF safe?

eBPF has access to the kernel, and improper use can lead to catastrophic results. Thus eBPF leverages multiple layers of security to ensure safety.

- Required privileges - by default, only eBPF programs running in privileged mode can be loaded into the kernel. If unprivileged mode is enabled, then eBPF gets loaded with limited functionality.
- Verifier - All programs must pass through the eBPF verifier to ensure the safety of the program itself.
- Hardening - the eBPF program is hardened by making it read-only. Also, guards are put against Spectre attack and JIT spraying attacks.
- Abstracted Runtime Context - eBPF programs cannot access arbitrary kernel memory directly or randomly modify data structures in the kernel.

Why eBPF?

Araali chose to included eBPF (in addition to other technology) in its toolkit as it allowed us to provide security in a way that is

1) Non-Disruptive - eBPF offers a network tap and guarantees against kernel crashes.
2) Performant - eBPF does not copy packets to userspace and implement JIT (just in time) compilation.
3) Secure - eBPF has added strick security features to prevent it from becomes an attack vectors for the adversaries
4) Programmability - eBPF allows us to reprogram the Linux kernel behavior without requiring changes to kernel source code or load a kernel module. Programmability gives us the option to keep adding new features and functionalities over time.
5) Multi taps allowed - eBPF allows many network taps to be active at the same time. Even if the customer is using other modules, it will not impact Araali and vice versa.

What are some limitations of using eBPF?

1) eBPF requires a fairly recent kernel (v4.9 or later)
2) Does not work where Araali does not have access to the kernel (e.g., lambda) - Araali has another operation mode in user space to cover lambdas