03/06/2022

Uptycs

eBPF and Linux Container Security

404 Wyman Street
Suite 357
Waltham, MA 02451

www.uptycs.com

Quick Overview

Key eBPF Concepts

eBPF (enhanced Berkeley Packet Filter) is a Linux kernel technology that offers a powerful and stable method of observing the Linux kernel. The eBPF sensor is like having a VM in the kernel that can safely run hooks (i.e. programs) for filtering data like network events, system calls, packets, and more. Many organizations have adopted eBPF at scale due to guaranteed stability, benefits of working directly in the kernel, and potential savings when factoring in the compute process for gathering telemetry on Linux servers and containers. eBPF is a safe way of interacting with the Linux kernel and a preferred alternative to plugging into the auditd framework.

Core benefits:

- **Speed and performance.** eBPF can move packet processing from the kernel-space and into the user-space. eBPF is also a just-in-time (JIT) compiler. After the bytecode is compiled, eBPF is invoked rather than a new interpretation of the bytecode for every method.
- **Low intrusiveness.** When leveraged as a debugger, eBPF doesn't need to stop a program to observe its state.
- **Security.** Programs are effectively sandboxed, meaning kernel source code remains protected and unchanged. The verification step ensures that resources don't get choked up with programs that run infinite loops.
- **Convenience.** It's less work to create code that hooks kernel functions than it is to build and maintain kernel modules.
- **Unified tracing.** eBPF gives you a single, powerful, and accessible framework for tracing processes. This increases visibility and security.
- **Programmability.** Using eBPF helps increase the feature-richness of an environment without adding additional layers. Likewise, since code is run directly in the kernel, it's possible to store data between eBPF events instead of dumping it like other tracers do.

A new-era of sensors

Incorporating eBPF into osquery gives in-depth 24/7 monitoring of servers and containers, this is especially important for cloud workloads in production. Combined with cloud automation, eBPF offers real-time security observability, speed, and convenience for monitoring extremely high-volume event data.

For osquery, eBPF is a preferred alternative to the auditd framework. The biggest difference between eBPF and auditd is that, with auditd, an agent can get bogged down or cause conflicts if multiple agents are pulling telemetry. eBPF does not run into these types of conflicts. eBPF has major advantages for assets that have agents or logging frameworks that rely on consuming events from auditd. Auditd can only have one one consumer of events, so using eBPF with osquery removes that resource constraint. eBPF is also less invasive, purely operating in a read-only mode and it runs close to the kernel, so it is more efficient.

Additionally, eBPF brings about context that auditd doesn't have about containers. The auditd framework can pull telemetry from containers but struggles with associating these events to namespaces or cgroups used to isolate processes across different containers. With eBPF, Uptycs correlates these events across your containers to provide a unified solution for monitoring container workloads.

eBPF at scale using Uptycs: Uptycs enhancements

With eBPF, Uptycs inserts probes into the kernel to monitor events of interest to us. This happens when osquery starts up and passes information back to the userland osquery process. This greatly reduces the resource utilization needed for in-depth security monitoring. eBPF is easily configured for this process and does not create any delays in deployment.

Compatibility: Uptycs also maps kernel memory to achieve compatibility across different Linux kernel versions i.e. compile ebpf program(s) once and use it quickly and easily across various kernel versions.

Detections: The telemetry we gather includes the ancestor list, showing who is the parent of that process and the related ancestry. This forms a powerful detections framework for tracing processes and forming event alerts, differing from the open-source osquery framework that requires multiple JOINs and correlating the PID with the process table to implement a container detections framework.