Efficient Audit Logging with eBPF

Rohit Aich
PhD Student, Secure Systems Lab

Stony Brook University
Computer Science

MOTIVATION

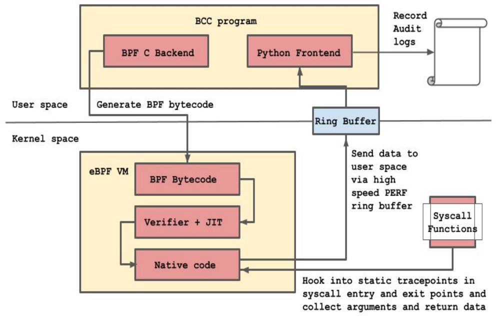
1. Large enterprises continue to be marred by stealthy and long-term cyber-attacks, commonly known as Advanced Persistent Threats (APTs).
2. The only way to detect and prevent such attacks is forensic analyses. The system audit logs provide crucial information for such analyses.
3. The existing approaches of logging system audit data involves installing one or more kernel modules, hence difficult to deploy and maintain.
4. Existing approaches like Linux Audit Daemon suffer from huge run-time and space overhead.
5. Also, system logs tend to be unnecessarily verbose, making it difficult to analyze them.

OUR APPROACH AND ITS KEY BENEFITS

- Introduces a lightweight audit logger written in BPF C and Python.
- Simple installation and deployment techniques, does not require to modify or rebuild the kernel.
- Leverages eBPF technologies to hook into system calls at predefined static tracepoints.
- Traces all system call arguments and return values and sends to userspace by a high-performance ring buffer.
- The system is absolutely lossless and thread-safe.

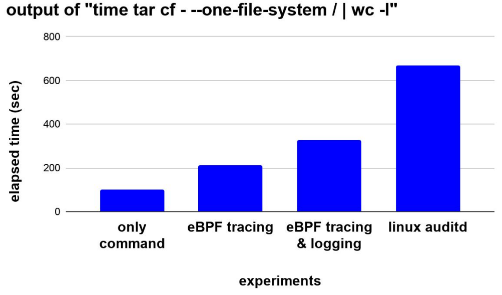
What is eBPF?

1. eBPF (Extended Berkeley Packet Filters) is a technology that can run sandboxed programs inside the kernel.
2. Has a register based VM using a custom 64 bit RISC instruction set capable of running Just-in-Time native-compiled "BPF programs".
3. Programs are event-driven; run when the kernel or an application passes a certain hook point.
4. No need to change kernel source code or load new modules.
5. Pre-defined hooks include system calls, network events, and several others.



System Architecture and Workflow

Comparison of runtime with Linux Auditd



PERFORMANCE & OPTIMIZATION

1. So far, our system supports the most frequent 25 system calls. As of now, the hooking and tracing by the eBPF system introduces a small run-time overhead.
2. The graph shows the runtime comparison of our system with the Linux Audit Daemon, with an experimental command.
3. We are currently trying to optimize the system to provide better run-time and storage usage performance.
4. We are building a cache-based filter inside our eBPF system to prevent redundant entries from getting logged.