Extended Berkeley Packet Filter for Intrusion Detection Implementations

COMP4906 Honours Thesis Proposal

by

William Findlay

November 6, 2019

Under the supervision of Dr. Anil Somayaji
Carleton University

Abstract

testificate

Acknowledgments

Contents

1 Introduction and Motivation **1**

2 Background **2**

2.1 Extended Berkeley Packet Filter and System Introspection 2

2.1.1 Linux Tracing Superpowers 3

2.1.2 The Verifier: The Good, the Bad, and the Ugly 3

2.1.3 eBPF Programs 3

2.1.4 XDP Programs 3

2.1.5 Other System Introspection Techniques 3

2.2 Intrusion Detection 3

2.3 Process Homeostasis 3

3 Implementing ebpH **3**

4 Methodology **3**

References **4**

Appendix Testificate **5**

List of Figures

2.1 Basic topology of eBPF with respect to userland and the kernel 3

List of Tables

List of Listings

1 Introduction and Motivation

As our computer systems grow increasingly complex, so too does it become more and more difficult to gauge precisely what they are doing at any given moment. Modern computers are often running hundreds, if not thousands of processes at any given time, the vast majority of which are running silently in the background. As a result, users often have a very limited notion of what exactly is happening on their systems, especially beyond that which they can actually see on their screens. An unfortunate corollary to this observation is that users *also* have no way of knowing whether their system may be *misbehaving* at a given moment, whether due to a malicious actor, buggy software, or simply some unfortunate combination of circumstances.

Recently, a lot of work has been done to help bridge this gap between system state and visibility, particularly through the introduction of powerful new tools such as *Extended Berkeley Packet Filter* (eBPF). Introduced to the Linux Kernel in a 2013 RFC and subsequent kernel patch [1], eBPF offers a promising interface for kernel introspection, particularly given its scope and unprecedented level of safety therein; although eBPF can examine any data structure or function in the kernel through the instrumentation of tracepoints, its safety is guaranteed via a bytecode verifier. What this means in practice is that we effectively have unlimited, highly performant, production-safe system introspection capabilities that can be used to monitor as much or as little system state as we desire.

Certainly, eBPF offers unprecedented system state visibility, but this is only scratching the surface of what this technology is capable of. With limitless tracing capabilities, we can construct powerful applications to enhance system security, stability, and performance. In theory, these applications can perform much of their work autonomously in the background, but are equally capable of functioning in a more interactive role, keeping the end user informed about changes in system state, particularly if these changes in state are undesired. To that end, I propose *ebpH* (*Extended Berkeley Process Homeostasis*), an intrusion detection system based entirely on eBPF that monitors process state in the form of system call sequences. By building and maintaining per-executable behavior profiles, ebpH can dynamically detect when processes are behaving outside of the status quo, and notify the user so that they can understand exactly what is going on.

A prototype of ebpH has been written using the Python interface provided by the BPF Compiler Collection [2], and preliminary tests show that it is capable of monitoring system state under moderate to heavy workloads with negligible overhead. What's more, zero kernel panics occurred during ebpH's development and early testing, which simply would not have

been possible without the safety guarantees that eBPF provides. The rest of this proposal will cover the necessary background material required to understand ebpH, describe several aspects of its implementation, including the many findings and pitfalls encountered along the way, and discuss the planned methodology for testing and iterating on this prototype going forward.

2 Background

While my work is primarily focused on the use of eBPF for maintaining system security and stability, my working prototype for ebpH borrows heavily from Anil Somayaji's *pH* or *Process Homeostais* [3], an anomaly-based intrusion detection and response system written as a patch for Linux Kernel 2.2. As such, in the following sections I will attempt to provide some background on eBPF, intrusion detection, and the design of the original pH system.

2.1 Extended Berkeley Packet Filter and System Introspection

System introspection is hardly a novel concept. Developers have been designing kernel modules and in-kernel APIs for this express purpose for decades. Some prominent examples include:

- SystemTap [4], which employs a variety of backends, but predominantly constructs loadable kernel modules
- The ptrace system call, used by applications such as strace for system call tracing

Extended Berkeley Packet Filter is

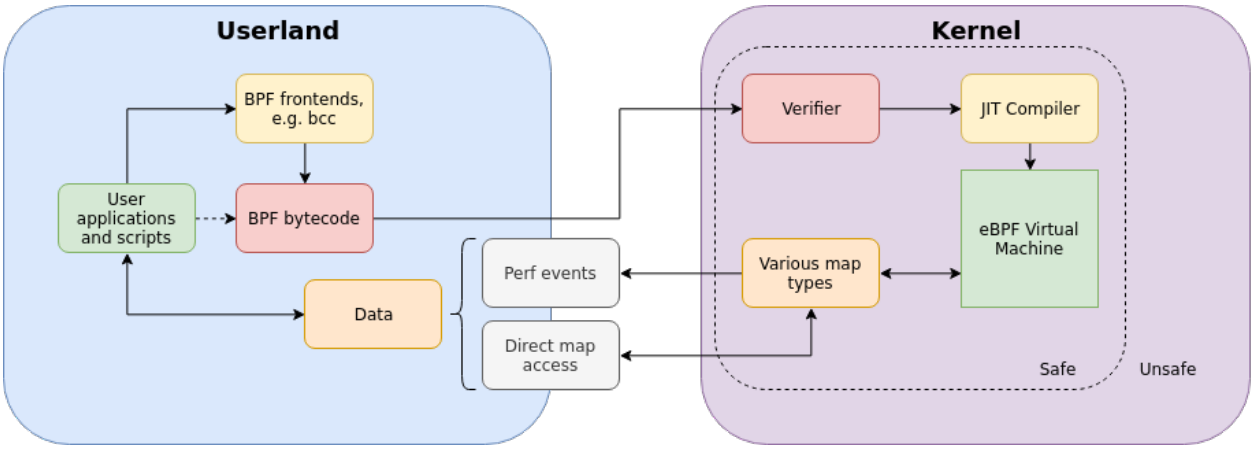

Figure 2.1: Basic topology of eBPF with respect to userland and the kernel. Note the bidirectional nature of dataflow between userspace and kernelspace using maps.

2.1.1 Linux Tracing Superpowers

2.1.2 The Verifier: The Good, the Bad, and the Ugly

2.1.3 eBPF Programs

2.1.4 XDP Programs

2.1.5 Other System Introspection Techniques

2.2 Intrusion Detection

2.3 Process Homeostasis

3 Implementing ebpH

4 Methodology

References

[1] A. Starovoitov, "Tracing filters with bpf," The Linux Foundation, RFC 0/5, Dec. 2013. [Online]. Available: https://lkml.org/lkml/2013/12/2/1066.

[2] *Iovisor/bcc*, Oct. 2019. [Online]. Available: https://github.com/iovisor/bcc.

[3] A. B. Somayaji, "Operating system stability and security through process homeostasis," PhD thesis, Anil Somayaji, 2002. [Online]. Available: https://people.scs.carleton.ca/~soma/pubs/soma-diss.pdf.

[4] *Understanding how systemtap works red hat enterprise linux 5*. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/systemtap_beginners_guide/understanding-how-systemtap-works.

[5] S. Goldstein, "The next linux superpower: Ebpf primer," USENIX SRECon16 Europe, Jul. 2016. [Online]. Available: https://www.usenix.org/conference/srecon16europe/program/presentation/goldshtein-ebpf-primer.

Appendix A Testificate