

bpfbbox: Simple Precise Process Confinement with eBPF

William Findlay

Anil Somayaji

David Barrera

Carleton University

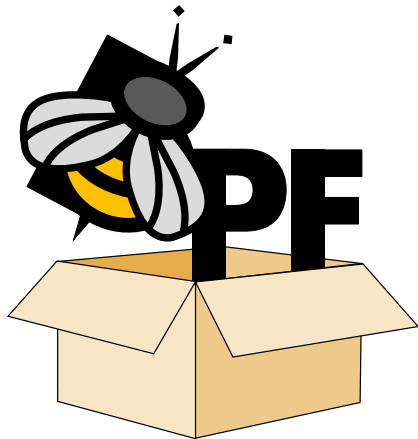
will@ccsl.carleton.ca

November 9, 2020



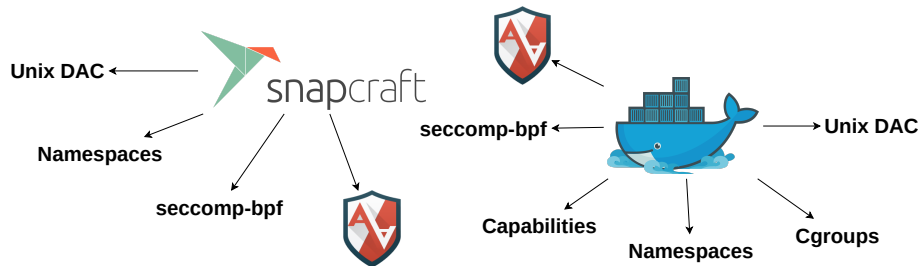
bpfbbox at a Glance

- ▶ bpfbbox is a novel **process confinement mechanism** for Linux
 - ▶ Using a new Linux technology called eBPF
- ▶ Users write per-application policy in a **simple policy language**
- ▶ Policy is enforced by attaching **eBPF programs** to **LSM hooks**
 - ▶ Integrates cross-layer state into policy decisions



Motivation

- ▶ Existing process confinement mechanisms are **complex**



- ▶ Existing process confinement mechanisms are **difficult to use**



- ▶ Can we do any better?

eBPF Changes the Game

eBPF enables:

- ▶ Fine-grained system introspection
- ▶ Integration of **cross-layer state** with policy enforcement
- ▶ Rapid prototyping
- ▶ Safe production deployment of new security solutions

We have an opportunity to **rethink process confinement** from the ground up.

eBPF in the Beginning

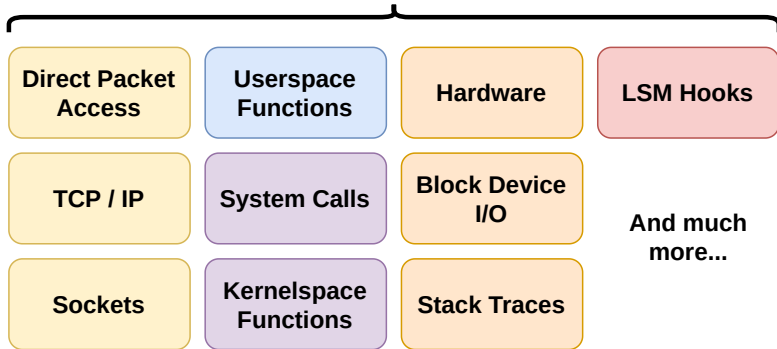
eBPF \equiv **E**xtended **B**erkley **P**acket **F**ilter

- ▶ But it has little to do with Berkley, packets, or filtering nowadays
- ▶ The name BPF is preserved for historical reasons

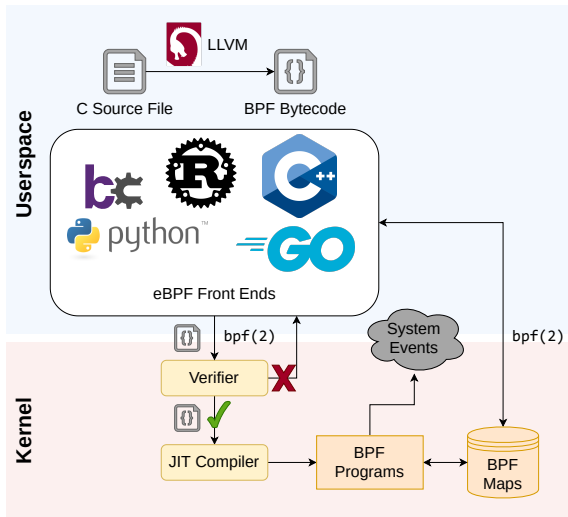
So then **what is eBPF?**

- ▶ A major re-write of the Linux BPF engine
 - ▶ Alexei Starovoitov and Daniel Borkmann
- ▶ Merged into the Linux kernel in 2014
- ▶ The original goal was fine-grained, cross-layer **system introspection**

What Can eBPF Do?



How eBPF Works



eBPF in 2020

eBPF is now **more than just an observability tool**.

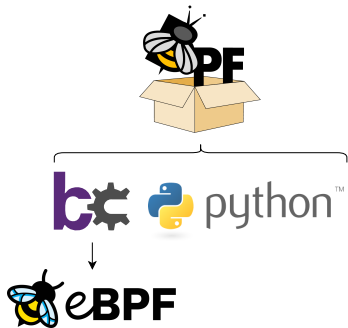
- ▶ eBPF provides a **safe, efficient, and flexible** way for privileged users to extend the kernel
- ▶ eBPF turns Linux into a **programmable kernel**

Linux 5.7 → KRSI (**K**ernel **R**untime **S**ecurity **I**strumentation)

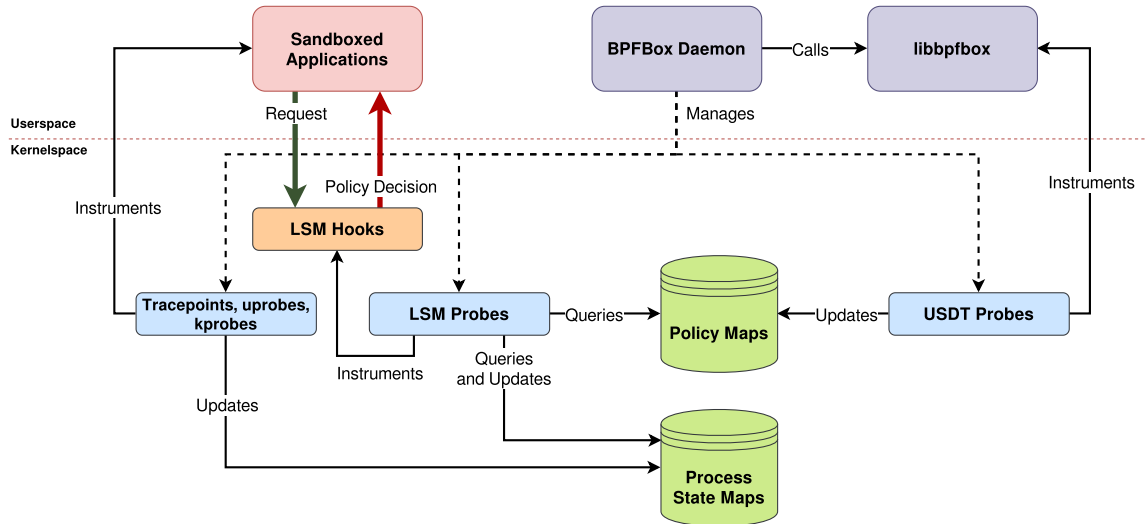
- ▶ Attach BPF programs to LSM hooks
- ▶ Make security decisions and generate audit logs with eBPF

bpffbox Implementation

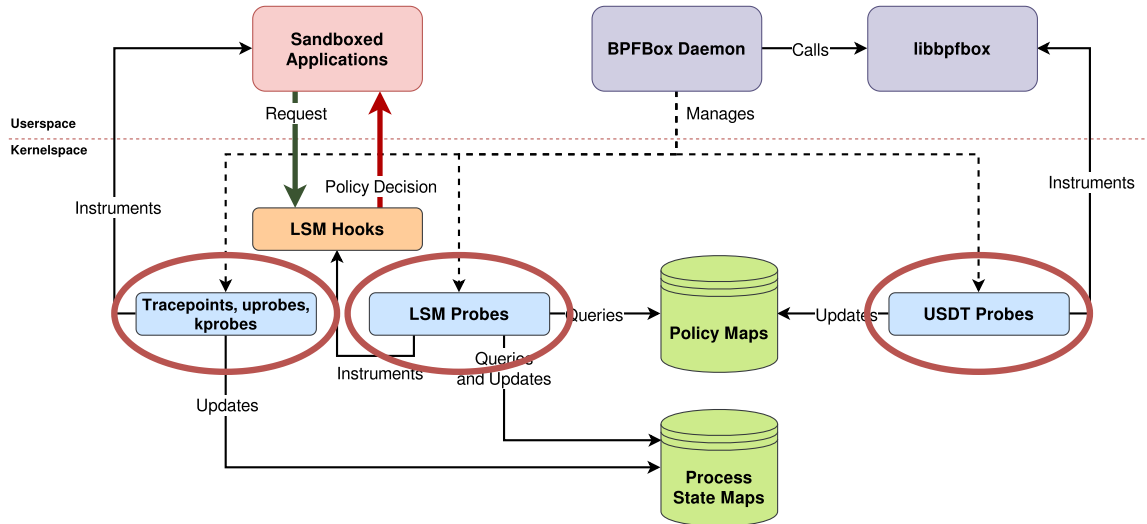
- ▶ Userspace daemon using the Python3 bcc framework
- ▶ Kernel space components are all written in eBPF
 - ▶ LSM probes (KRSI), kprobes, uprobes
 - ▶ Under 2000 source lines of kernel space code
- ▶ Thanks to eBPF, bpffbox is **light-weight**, **flexible**, and **production-safe**
 - ▶ Works out of the box on any vanilla Linux kernel ≥ 5.8



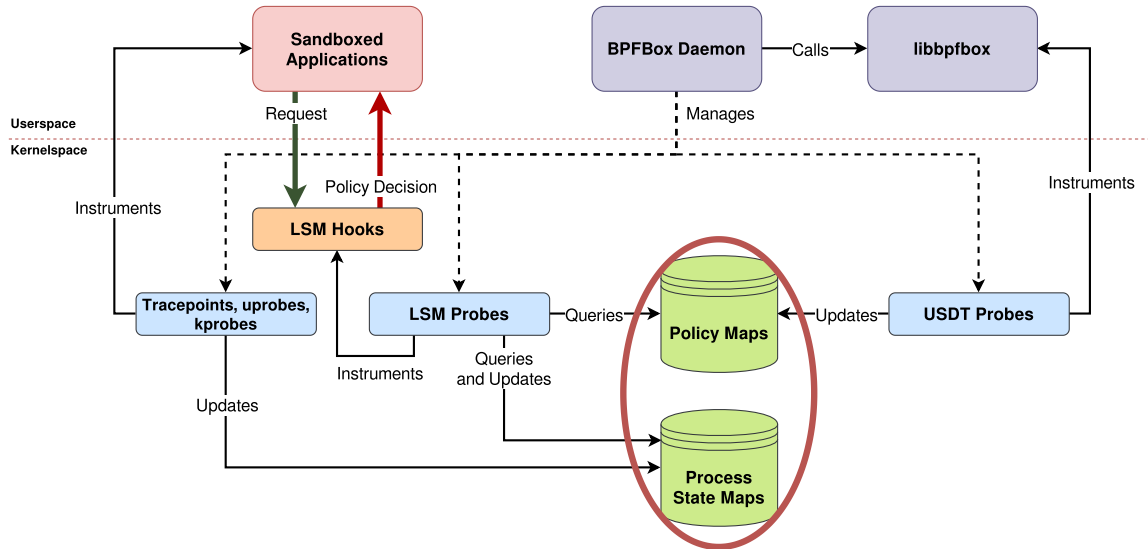
bpfbox Architecture



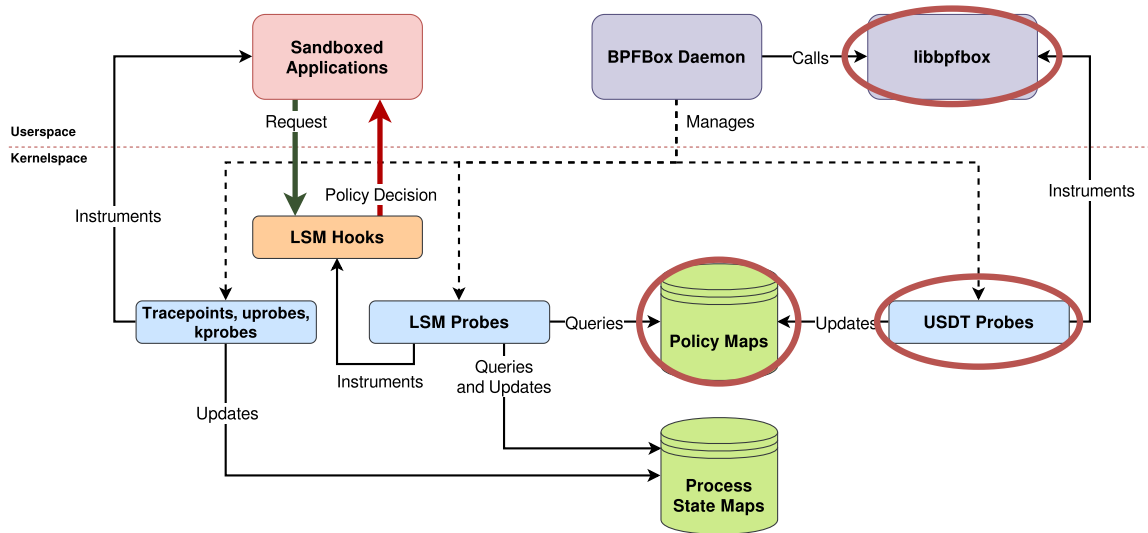
bpfbox Architecture



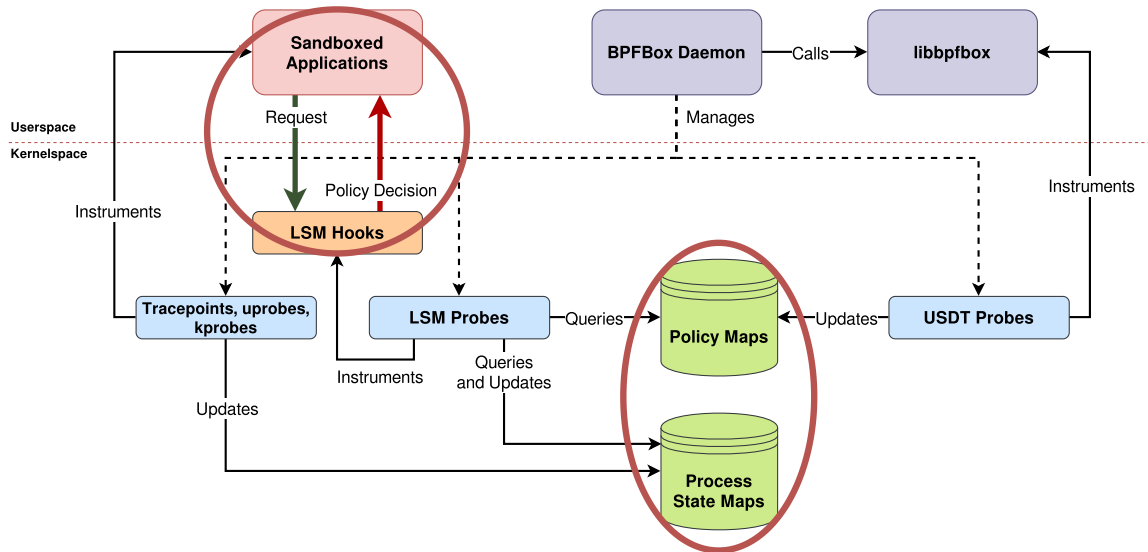
bpfbox Architecture



bpfbox Architecture



bpfbox Architecture



Policy Design Goals

1. Simplicity

- ▶ Policy should be simple enough for ad hoc confinement

2. Application transparency

- ▶ Policy should not require changes to the confined application

3. Flexibility

- ▶ Policy should offer optional layers of granularity

4. Security

- ▶ Policy should follow the principle of least privilege
- ▶ It should be difficult to write an insecure policy

Rules and Directives

Rules specify access to system objects:

- ▶ `fs(file, access)`
- ▶ `net(socket, access)`
- ▶ `signal(prog, sig)`
- ▶ etc.

Directives augment blocks of rules:

- ▶ `#[directive]` syntax
- ▶ Specify **actions to be taken** on a block of rules
- ▶ Add **additional context** to a block of rules

Taints and Transitions

- ▶ `#[taint]` → Start confinement
- ▶ `#[transition]` → Switch profiles on `execve`

```
#![profile "/bin/mywebdaemon"]

#[taint] {
    net(inet, any)
    net(inet6, any)
}

/* ... */

#[transition] {
    fs("/bin/myhelper", getattr|read|exec)
}
```

Policy at the Function Call Level

- ▶ `#[func "foo"]` → Apply rules only within a call to `foo()`
- ▶ `#[kfunc "foo"]` → Same thing, but for kernel functions

```
#[profile "/sbin/mylogin"]

#[func "check_password"]
#[allow] {
    fs("/etc/passwd", read)
    fs("/etc/shadow", read)
}

#[func "add_user"]
#[allow] {
    fs("/etc/passwd", read|append)
    fs("/etc/shadow", read|append)
}

/* ... */
```

Performance Evaluation

Methodology

- ▶ Phoronix Test Suite OSBench
 - ▶ Measures basic OS functionality
 - ▶ (spawning processes, memory allocations, etc.)
- ▶ Phoronix Test Suite Apache
 - ▶ Benchmark Apache `httpd` packets per second
- ▶ Kernel compilation benchmarks
 - ▶ Measure Linux kernel compilation performance
 - ▶ Heavy workload, spawning lots of processes

Performance Evaluation

Methodology

Two modes of operation for each test.

- ▶ Passive mode
 - ▶ bpfbox and AppArmor instrument hooks, but do not enforce or audit
 - ▶ Test lowest possible overhead
- ▶ Complaining mode
 - ▶ bpfbox and AppArmor complain about (log) every security-sensitive operation
 - ▶ Test worst case overhead

Performance Evaluation

Results

- ▶ Phoronix OSBench
 - ▶ Passive: bpfbox is **roughly equivalent** to AppArmor
 - ▶ Complaining: bpfbox performs **significantly better** than AppArmor
- ▶ Phoronix Apache
 - ▶ bpfbox and AppArmor are **roughly equivalent**
- ▶ Kernel compilation
 - ▶ Passive: bpfbox is **roughly equivalent** to AppArmor
 - ▶ Complaining: bpfbox performs **better in kernelspace** overhead and **worse in userspace** overhead

Contributions

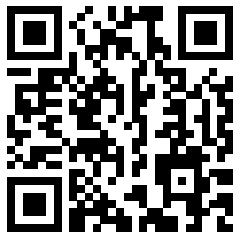
- ▶ First **policy enforcement engine** written in **eBPF**
- ▶ Integration of **userspace** and **kernelspace** state with **LSM layer enforcement**
- ▶ A simple policy language for **ad hoc process confinement**
 - ▶ But with optional complexity for **fine-grained protection**

Acknowledgements

Special thanks to:

- ▶ **Alexei Starovoitov** and **Daniel Borkmann** (creators of eBPF)
- ▶ **K.P. Singh** (creator of KRSI)
- ▶ Fellow **bcc contributors** (an awesome eBPF framework)
- ▶ Anonymous **CCSW'2020 reviewers** (valuable feedback)

This work was supported by NSERC through a Discovery Grant.



github.com/willfindlay/bpfbox

Check out the project on GitHub!