

# **bpfbox: Simple Precise Process Confinement in eBPF**

**William Findlay**   Anil Somayaji   David Barrera

Carleton University  
[will@ccsl.carleton.ca](mailto:will@ccsl.carleton.ca)

October 19, 2020

bpfbox logo here

# Outline of Talk

What is eBPF?

Motivation

bpfbox Implementation

bpfbox Policy

Performance Evaluation

Conclusion

# What is eBPF?

# eBPF in the Beginning

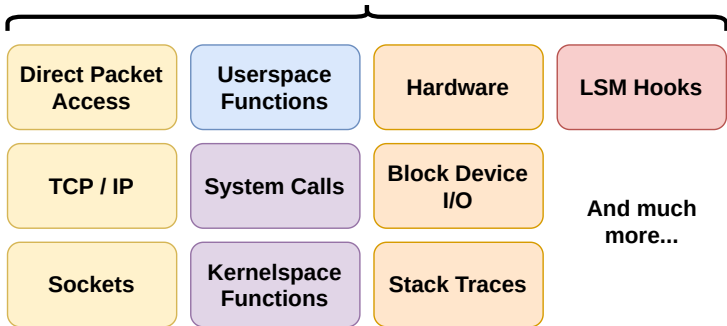
eBPF  $\equiv$  **E**xtended **B**erkley **P**acket **F**ilter...

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

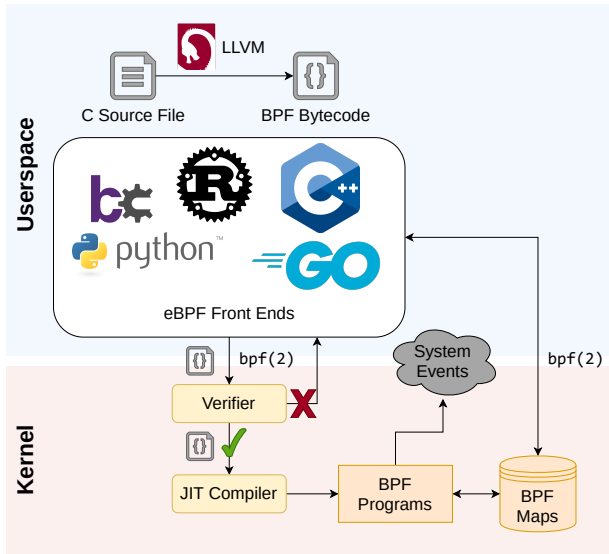
So then **what is eBPF?**

- ▶ A major re-write of the Linux BPF engine
  - ▶ Alexei Starovoitov and Daniel Borkmann
- ▶ Merged into the Linux kernel in 2014
- ▶ The point 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)

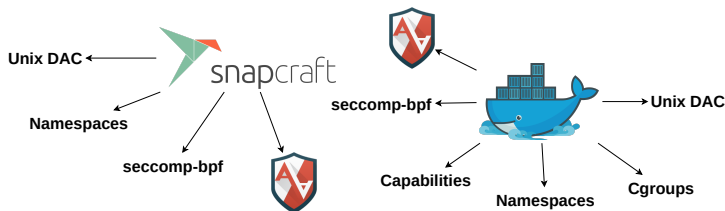
- ▶ Attach BPF programs to LSM hooks
- ▶ Make security decisions and generate audit logs with eBPF



# Motivation

# The Status Quo

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



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



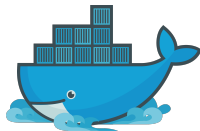
- ▶ Can we do any better?

# Stakeholders as Policy Authors

- ▶ Security experts define the policy



- ▶ Application authors and **packagers** define the policy



- ▶ End users define the policy

???

# eBPF Changes the Game

eBPF enables:

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

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

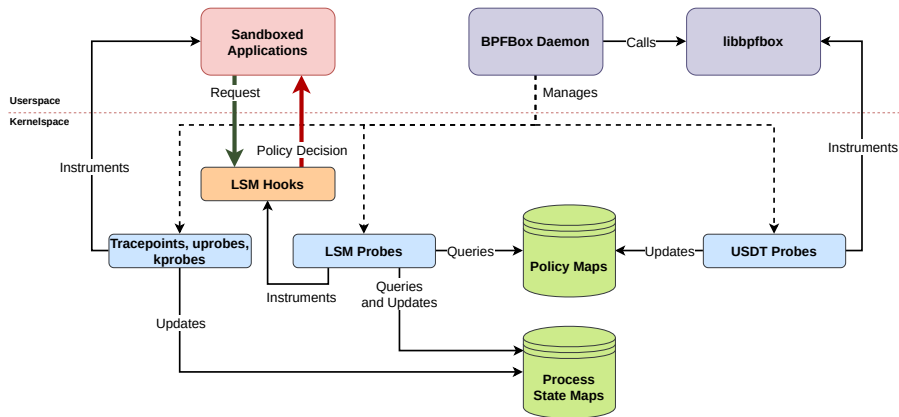
# **bpfbbox**

# **Implementation**

# bpffbox Implementation

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

# bpffbox Architecture



# **bpfbox Policy**



# 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

# Policy at the Function Call Level

```
#!/[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)
}

/* ... */
```

# bpfbox vs. Snap

- ▶ Snap provides TODO

See our paper for a detailed case study of Apache httpd policy.

# **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

# Results

- ▶ Phoronix OSBench

- ▶ Average case: bpfbox is **roughly equivalent** to AppArmor
- ▶ Worst case: bpfbox performs **significantly better** than AppArmor

- ▶ Phoronix Apache

- ▶ bpfbox and AppArmor are **roughly equivalent**

- ▶ Kernel compilation

- ▶ Average case: bpfbox is **roughly equivalent** to AppArmor
- ▶ Worst case: bpfbox performs **better in kernelspace** overhead and **worse in userspace** overhead

# Conclusion

# Acknowledgements

Special thanks to:

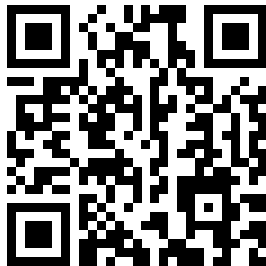
- ▶ **Alexei Starovoitov** and **Daniel Borkmann** (creators of eBPF)
- ▶ **K.P. Singh** (creator of KRSI)
- ▶ Fellow **bcc contributors** (an awesome eBPF framework)
- ▶ Anonymous **CCSW'2020 reviewers** (valuable feedback)

This work was supported by NSERC through a Discovery Grant.



# 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**



[github.com/willfindlay/bpfbox](https://github.com/willfindlay/bpfbox)

Check out the project on GitHub!