

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

**William Findlay**   Anil Somayaji   David Barrera

Carleton University  
`will@ccsl.carleton.ca`

October 18, 2020

# Outline of Talk

What is eBPF?

Motivation

bpfbox Architecture

bpfbox Policy

Performance Evaluation

Conclusion

# What is eBPF?

# eBPF in the Beginning

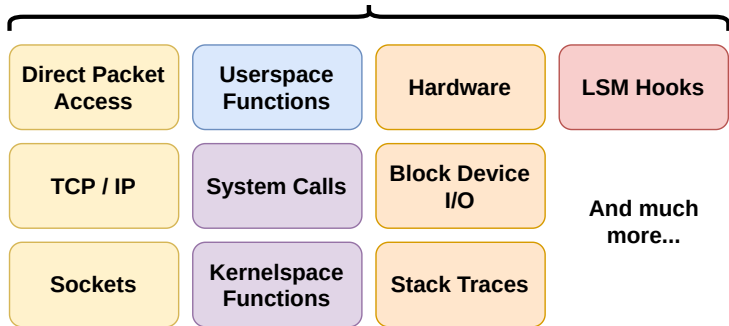
eBPF  $\equiv$  **E**xtended **B**erkley **P**acket **F**ilter...

- ▶ But it has little to do with Berkley, packets, or filtering nowadays

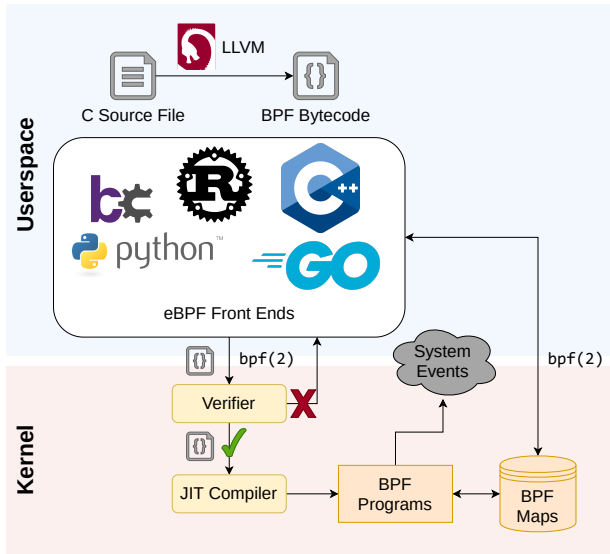
So then **what is eBPF?**

- ▶ A major re-write of the Linux BPF engine
  - ▶ Alexei Starovoitov and Daniel Borkman
- ▶ 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

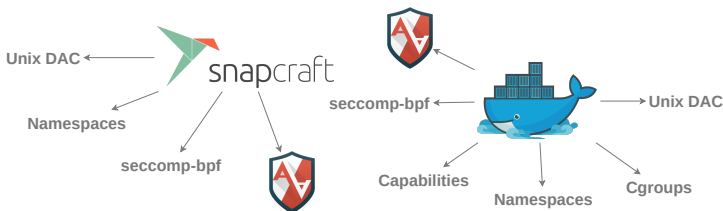
- ▶ eBPF as more than just an observability tool

# 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

TODO

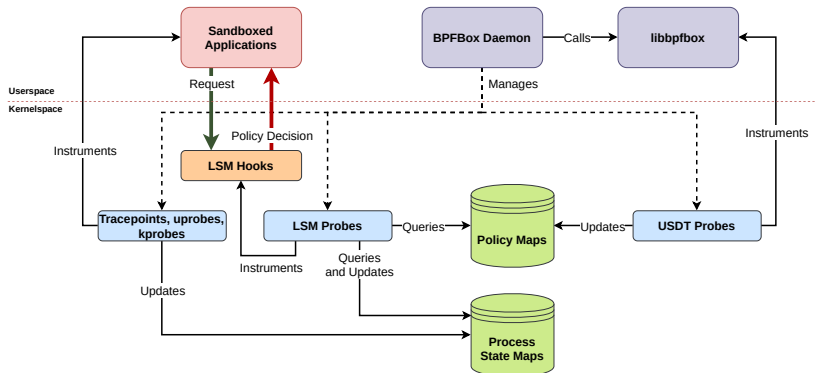
# **bpfbox**

# **Architecture**

# bpfbox Architecture

- ▶ TODO: Python3 bcc
- ▶ TODO: KRSI
- ▶ TODO: Lines of userspace code
- ▶ TODO: Lines of kernelspace code
- ▶ TODO: Compare w/ SELinux, AppArmor

# bpffbox Architecture



# **bpfbox Policy**

# Policy at the Function Call Level

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

```
#[func check_password]
```

```
#[func add_user]
```

```
#[allow] {  
    read("/etc/passwd")  
    read("/etc/shadow")  
}
```

```
#[func add_user]
```

```
#[allow] {  
    append("/etc/passwd")  
    append("/etc/shadow")  
}
```



# **Performance Evaluation**

# Performance

TODO

# Conclusion

# Acknowledgements

TODO

# Contributions

- ▶ First full policy enforcement engine written in eBPF
- ▶ Integration of **userspace** and **kernelspace** state with **LSM layer enforcement**