BPFContain[1]: Towards Secure and Usable Containers with eBPF

William Findlay

Carleton University
will@ccsl.carleton.ca

December 8, 2020

COMP5900I Final Project Presentation

[1] If you have alternative name suggestions, let me know!

Outline of this Talk

1. Containers and Security
2. BPFContain Policy
3. BPFContain Implementation
4. Conclusion

Containers and Security

Containers vs Full Virtualization

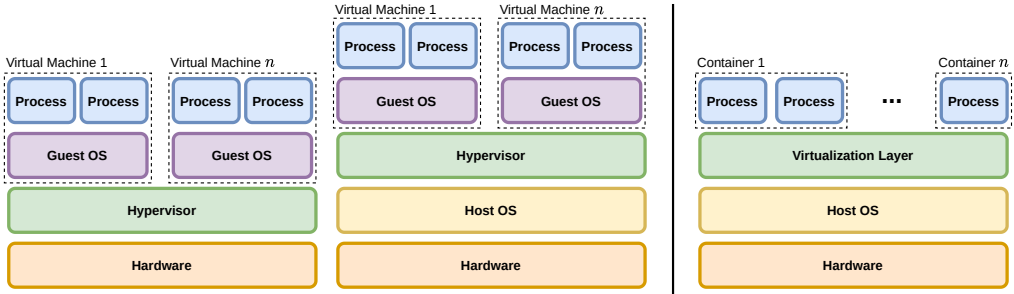


Figure based on a simpler figure in [2]

Containers vs Full Virtualization

Containers:

- Lighter weight
- Share the underlying host OS kernel, resources
- Managed directly by the host OS
- Weaker isolation guarantees

Full Virtualization:

- Heavier weight (especially Type II)
- Each virtual machine includes its own guest operating system
- Managed by a hypervisor (sometimes without a host OS)
- Stronger isolation guarantees

Containers: Goals

Virtualization.

- Give each container a *virtual view* of system resources
- Process tree, filesystems, networking stack, memory, etc.

Least-Privilege.

- *Restrict access* to sensitive system resources
- Prevent a compromised container from compromising the rest of the system

Composability.

- Composable micro-services
- Allow containers to interact with each other (in pre-defined ways)

Also dependency management, but we'll ignore that for now.

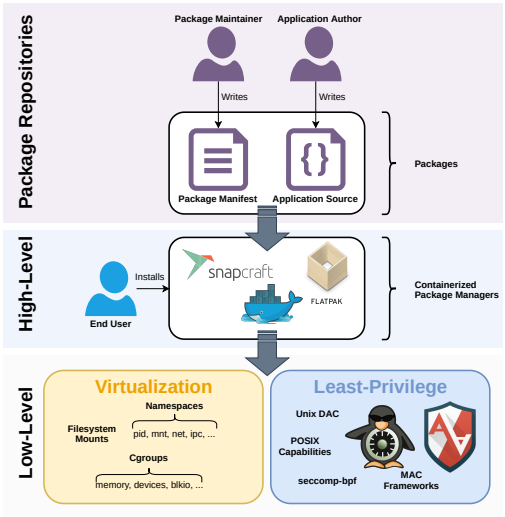
Containers: Complexity

Virtualization.

- Filesystem mounts
- Namespaces
- Cgroups

Least-privilege.

- Unix DAC
- Seccomp
- POSIX capabilities
- MAC LSMs (e.g. SELinux/AppArmor)



Containers: Security

Container security is the **primary barrier** to widespread adoption [2].

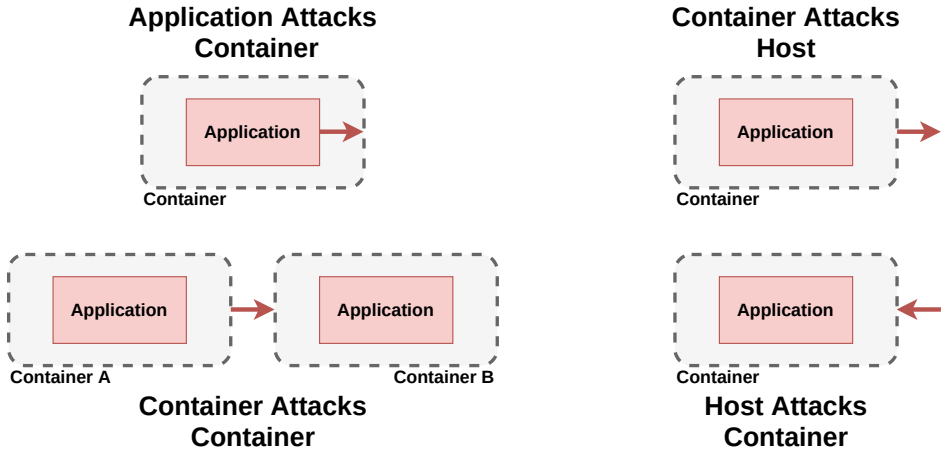
Container security is generally an **afterthought**.

- Containers share the underlying OS kernel
- Proper virtualization/confinement requires elevated privileges (handle with care)
- Without **full** support for underlying security mechanisms (e.g. MAC LSMs), we cannot achieve least-privilege

How can we **fix container security**?

- Build containers to be **secure from the ground up**
- That means **start with least-privilege**, make virtualization optional
- Write an LSM **specifically for containers** [2]

Containers: Threat Model



Containers: Threat Model

Escalation of privilege.

- Escape confinement
- Collusion attacks with other processes/containers

Tampering.

- Mess with other containers
- Mess with the host

Information disclosure.

- Reveal sensitive/secret information

Denial of service.

- Kill processes
- Disable interfaces
- Consume resources

BPFContain's Mission Statement

Help defenders by providing **simple, secure containers** with policy that is **easy to customize**.

BPFContain Policy

Policy Design Goals

1. Support the same basic functionality as Docker, Snap, etc.
   - Confinement + virtualization (currently confinement only)
   - Have a way for containers to talk to each other (composability)
2. Policy should be high-level, human-readable.
   - Motto: "Be friendlier than bpfbox."
3. No security knowledge required to read/write basic policy.
   - If I want to stop behaviour B_i, I shouldn't **have to** tune $\{B_1, ..., B_n\}$
   - But, I should **be able to** if necessary
4. Least-privilege by default.
   - Default-deny policy
   - No need to "fall back to no security" like Docker, Snap, etc.

Policy Language

Policy language is written in YAML [1].

- Simple, human-readable

Rights and **restrictions**.

- Rights ≡ What the container is **allowed** to do.
- Restrictions ≡ What the container is **not allowed** to do.
- A restriction *always* overpowers a right.

Policy is **default-deny**.

- But you can use default-allow if you like
- Simple use cases, like blocking a specific feature

Example Policy: Discord

Here's a policy that might be shipped with a Discord container:

```
name: discord
command: /usr/bin/discord
rights:
  - filesystem /
  - filesystem /proc readonly
  - network
  - video
  - sound
```

It fits on one slide!

Example Policy: Discord

Suppose the user doesn't want Discord scanning `procfs`:

```
name: discord
command: /usr/bin/discord
rights:
  - filesystem /
  - network
  - video
  - sound
```

Just remove the access right!

Example Policy: Discord

What if there is no existing policy? (And you don't want to write one)

```
name: discord
command: /usr/bin/discord
default: allow
restrictions:
  - filesystem /proc
```

Mark it as default-allow and specify a restriction.

BPFContain Implementation

BPFContain Components

The **policy enforcement engine** is written in eBPF.

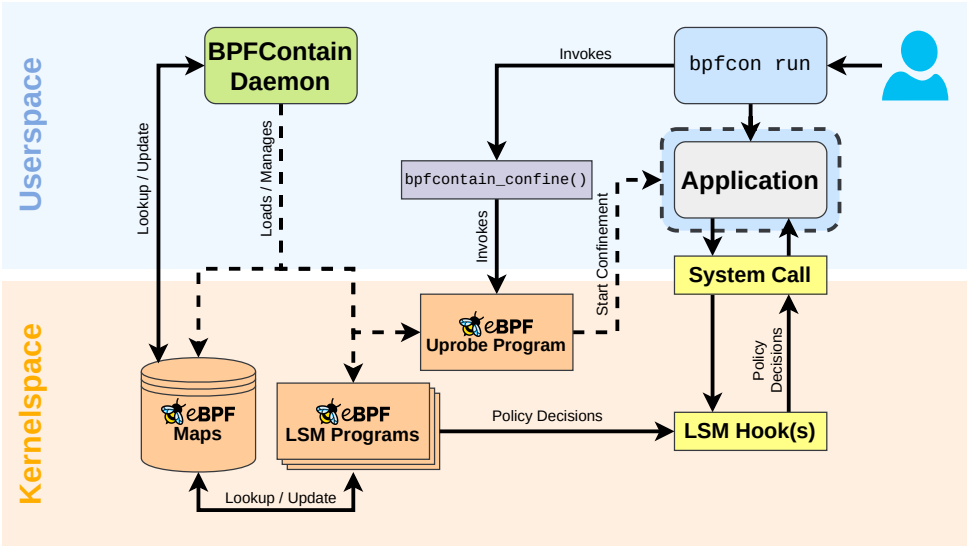
- Maps contain per-container policy
- eBPF programs attached to LSM hooks enforce policy
- Another eBPF program attached to a special library call associates a process group with a container ID

A **privileged userspace daemon** loads and manages BPF programs and maps.

The user executes containers using an **unprivileged wrapper application**.

- Launching a container requires **no additional privileges**
- You just run the container as an ordinary user

BPFContain Architecture



Conclusion

Future Work

Add virtualization to policy language.

- How to split things up?
- `mount` rules + `resource` rules?
- `namespace` rules + `cgroup` rules?

New eBPF helpers for virtualization.

- `bpf_enter_namespace()`
- `bpf_enter_cgroup()`
- `bpf_remount()` (in mount namespace only?)

Future Work

Get rid of the privileged daemon.

- We don't need it (just pin the BPF maps/programs and exit)
- `$ sudo bpfcon load-policy`
- `$ sudo bpfcon logs`

Global meta-policy.

- A policy about what policy is allowed
- E.g. "No default-allow policies on my system"
- E.g. "No IPC between any containers"

Contributions

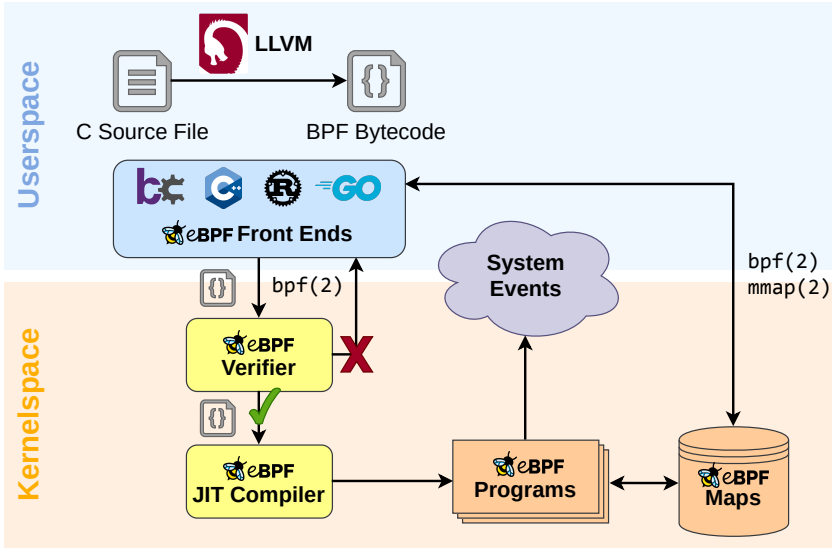
BPFContain can be framed in different ways:

1. A security-first approach to containers.
2. A simple approach to confinement.
3. A "container-specific" LSM implementation [2].

BPFContain helps defenders by providing simple, secure containers and easily customizable policy.

Backup Slides (eBPF 101)

eBPF In a Nutshell



Verifiable Safety

Limited instruction set.

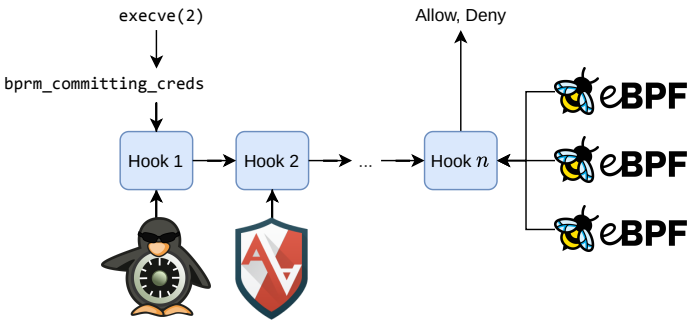
- 11 registers (10 general purpose)
- 114 instructions (vs 2000+ in x86)
- Access to a limited set of **kernel helpers** with `call` instruction

Restricted execution context.

- 512 byte stack limit
- Memory access must be bounds-checked
- No unbounded loops
- No back-edges in control flow
- **eBPF is not Turing complete**

KRSI: eBPF LSM Programs

- Attach one or more eBPF programs to a given LSM hook
- eBPF programs can then be used to make policy decisions
- Works co-operatively with other LSMs (SELinux, AppArmor, etc.)



References I

[1] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. *YAML Ain't Markup Language (YAML™) Version 1.2*. YAML specification. URL: https://yaml.org/spec/1.2/spec.html (visited on 11/29/2020).

[2] S. Sultan, I. Ahmad, and T. Dimitriou. "Container Security: Issues, Challenges, and the Road Ahead". In: *IEEE Access* 7 (2019), pp. 52976–52996. DOI: 10.1109/ACCESS.2019.2911732.