Container Security for MEC

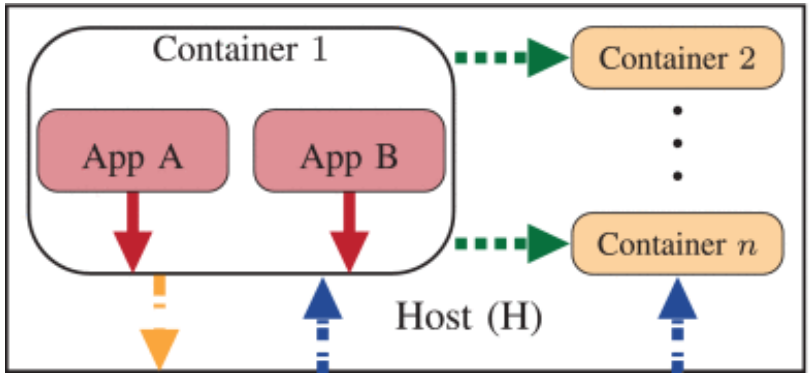
June 3rd, 2021

정수환
숭실대학교

1

Threat models

Container security – Threat models



Application attacks container (Use case I)

Container attacks other containers (Use case II)

Container attacks the host (Use case III)

Host attacks container (Use case IV)

- Use case I: Protecting a Container From Applications Inside It
- Use Case II: Inter-Container Protection
- Use Case III: Protecting the Host (And the Applications Inside it) From Containers
- Use Case IV: Protecting Containers from the Host

Use case I: Protecting a Container From Applications Inside It

Assumptions:

- Each application within a running container can be benign or malware.
- Applications cannot break access control policies if set
- Some applications might require root privileges (or parts of the full root access)

Attack vector:

- Image vulnerabilities (e.g., CVE-2014-6271 ShellShock exploit to attack the ***Bash*** to execute arbitrary commands or gain unauthorize access)
- Embedded malware
- Clear-text secrets stored inside the container

Use Case II: Inter-Container Protection

- **Assumptions:**
  - Containers are malicious
  - Applications inside are malicious, too
  - Host is not malicious
- **Attack vector:**
  - Poorly separated inter-container network traffic (e.g., DoS, ARP spoofing attack)
  - Container runtime vulnerable (e.g., malicious container can make change on another container resource)

Use Case III: Protecting the Host (And the Applications Inside it) From Containers

- **Assumptions:**
  - Container are malicious
  - Host is not malicious
- **Attack vector:**
  - Sharing host OS kernel leads to kernel exploit from container (e.g., CVE-2016-5195 Dirty Cow)
  - Resource accountability (i.e., DoS attack on CPU or Memory resource)
  - Host filesystem tampering (i.e., through mounting configuration defects to escape out of mount directory)

Use Case IV: Protecting Containers from the Host

- **Assumptions:**
  - Containers are not malicious
  - Host can be malicious (or have vulnerabilities)
- **Attack vector:**
  - Confidential information tampering or modifying (i.e., Host OS can control network devices, memory, storage and processors)
  - Unauthorized access to container data
  - Change behavior of container or application inside

Protection mechanisms

Software-Based Protection Mechanisms

- Linux Kernel Features (e.g., namespaces, Cgroup, Capabilities, Seccomp, etc.)
- Linux Security Modules (LSMs) (e.g., AppArmor, SELinux)
- Intrusion Detection System and Intrusion Protection System
- Container analysis (static, dynamic)

Hardware-Based Protection Mechanisms

- Trusted Execution Environment (TEE)
- TPM and vTPM (virtual TPM)
- Intel SGX

> *When your system has well-designed, the threats mostly come from the container images.*

2

Dockerfile Analysis for Vulnerable Scanning

Container image vulnerability scanning solutions

clair

A Container Image Security Analyzer by CoreOS

Trivy

Dagda

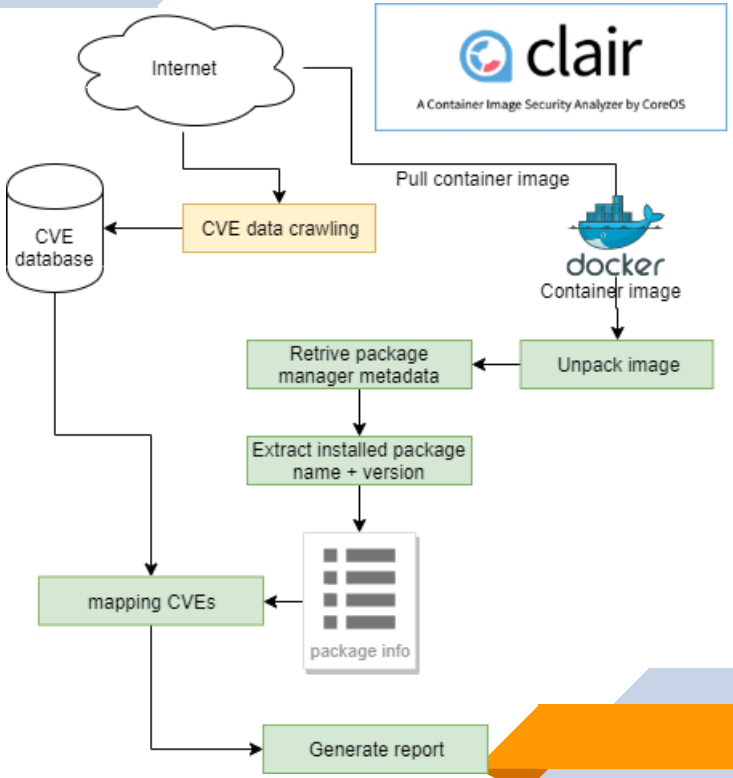
anchore

Docker security suite

Container image vulnerability scanning solutions

These tools do the followings:

- Collecting the CVEs data source;
- Extracting the installed packages
- Mapping CVEs
- Generating report



Container image vulnerability scanning solutions

Extract the installed packages based on:

- Package manager metadata, which includes information of installed package
  - For example, *dkpg* is a package manager of Debian distro. The packages metadata is inside: "*/var/lib/dpkg/status*".
- Programing language library manager
  - pip – a library manager for python
  - npm – a library manager for NodeJS

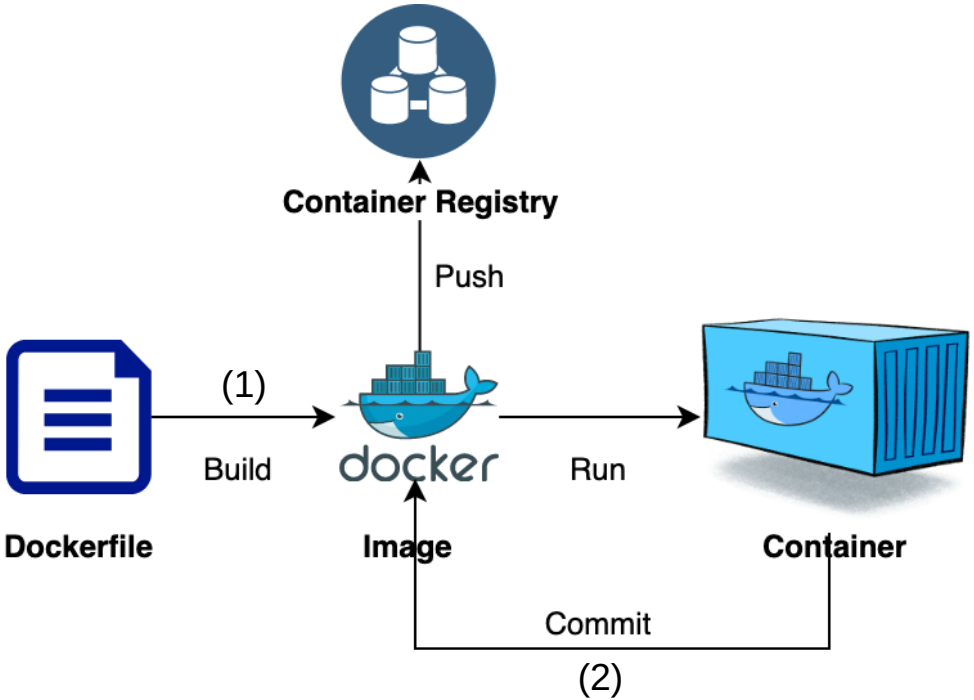
> How to scan **pre-compiled** software, **downloaded** from the internet or **copied** to the image?

Approach

How can we extract the information of pre-compiled, downloaded and added packages?

=> Can read the information in **Dockerfile.**

Docker container operation cycle



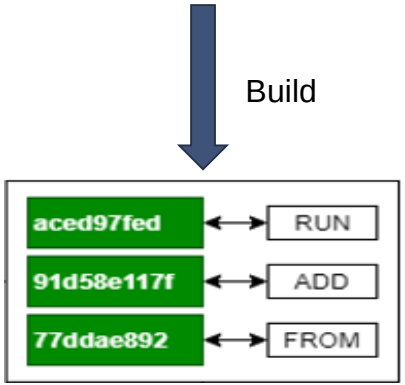
- A container image can be created by 2 way:
  - (1) Built from Dockerfile
  - (2) The change of running container
- Container image is stored in Container Registry (e.g., Dockerhub) using *push command*

Docker container operation cycle

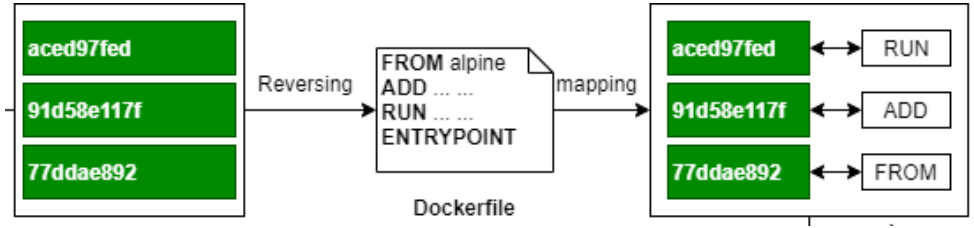
- Dockerfile: A script file written by developer for making a container image.
- There are some instructions (e.g., FROM, COPY, RUN, CMD, etc.) inside a Dockerfile which guide the image compiling system to build a container image.
- Normally, each Instruction will become an *image layer* of the container image.
- Each layer has an ID.

While storing in Container Registry, Dockerfile is not attached in a container image.

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



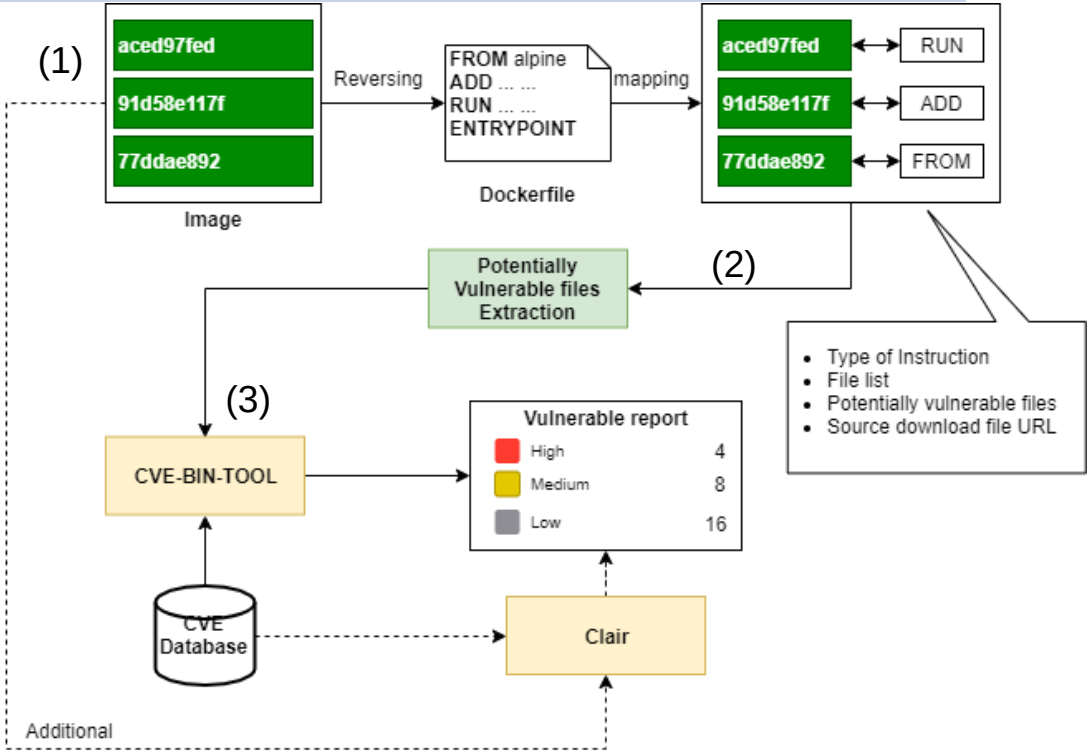
Dockerfile reversing



To get Dockerfile content out of the container image:

- (1): Extract “history” field from the manifest file.
- (2): Translate the history component to the instruction name.
- (3): Map the instruction name with the corresponding layer ID

DAVS: Dockerfile Analysis for Vulnerable Scanning



- (1) Reverse the Dockerfile from a container image
- (2) Extract Potentially Vulnerable Files (PVFs)
- (3) Scanning PVFs for vulnerable

DAVS: Extract Potentially Vulnerable Files

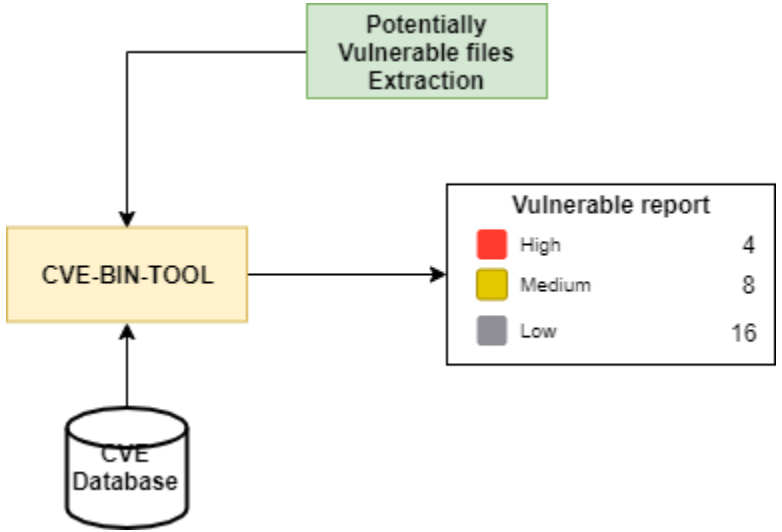
```
FROM ubuntu:18.04
ADD
file:8a9218592e5d736a05a1821a6dd38b205cdd8197c26a5aa33f6fc22fbfaa1c
4d in /
CMD ["bash"]
LABEL maintainer=phithon <root@leavesongs.com>
RUN /bin/sh -c apt-get update \
    && apt-get install -y autoconf automake build-essential […]\
    && wget -qO- https://www.ffmpeg.org/releases/ffmpeg-
2.8.4.tar.gz
    && cd /usr/src \
    && ./configure --pkg-config-flags="--static" --disable-yasm \
    && make \
    && make install \
    && rm -rf /usr/src/*
CMD ["ffmpeg"]
LABEL maintainer=phithon <root@leavesongs.com>
RUN /bin/sh -c set -ex \
    && apt-get update \
    && apt-get install -y --no-install-recommends php-cli \
    && rm -rf /var/lib/apt/lists/*
```

DAVS coverages

Clair coverages

Vulnerable scanning

- After analyze the Dockerfile, we can extract Potentially Vulnerable Files (PVFs)
- We leverage CVE-BIN-TOOL (a binary information extracting and scanning tool from Intel) to scan PVFs



Why don't we use CVE-Bin-tool for scanning all the files?

=> Because of performance problem

> *Static analysis is never enough for security*

3

Runtime Security

Introduction

Runtime security attacks happen when

Existing set of software systems (application services, databases, external APIs)

Application team has an expected path of execution and data flows.

Attacker finds an alternate but still permitted path of execution and data flow.

Runtime security is about

Enabling apps to run, and developers to move as fast as possible…

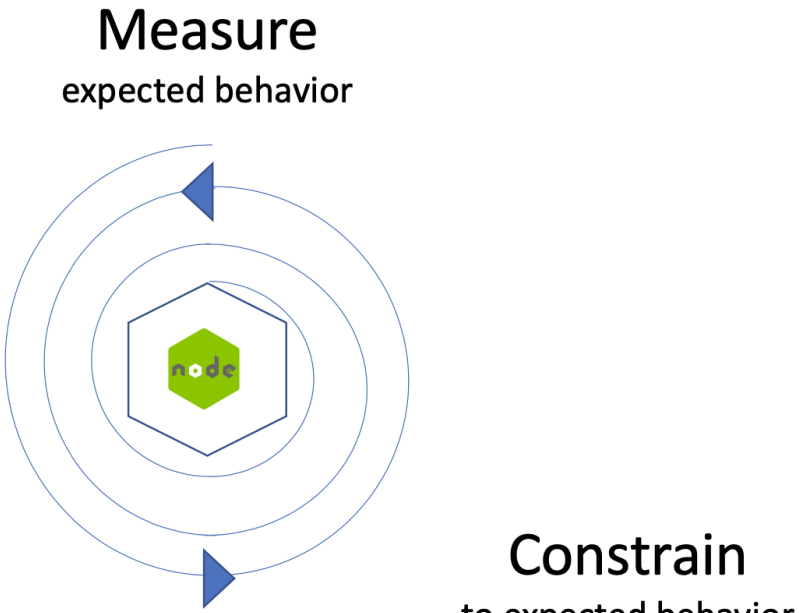
… while preventing execution and dataflow paths not intended by the app developers

Securing Microservices

- General Purpose OS leaves many degrees of freedom => many paths for exploit.
- What unique attributes of Kubernetes microservices can we leverage?
  - Single service per-container.
  - The process recognizes itself as PID 0.
  - Additional code run as sidecar containers.
  - Service code updated by deployment of new container.
  - Each service provides APIs.
  - Communication moving from IP address level to Socket level (Inter Process Communication).

Runtime Security

- Approach to Runtime Security
  - Monitor: what operations are normal
  - Constrain: enforce policies on the system
  - Measure: evaluate the system's security
  - Repeat



Berkeley Packet Filter

- What is BPF?
  - Highly efficient sandboxed virtual machine in the Linux Kernel
  - Making the Linux kernel (any distro) programmable at native execution speed
  - Original purpose of BPF is for Network Packet filtering.
- When Linux kernel version 3.18 was released in December 2014 it included the first implementation of extended BPF (eBPF)

Extended Berkeley Packet Filter (eBPF)

- Not only for networking subsystem
  - Tracing and Profiling: The ability to attach eBPF programs to trace application or kernel behavior
  - Observability & Monitoring: enables the collection of custom metrics and a wide range of sources for monitoring system with significantly reducing system overhead.
- eBPF uses a global data store that is called eBPF Maps for sharing of data:
  - Between kernel programs
  - Between kernel-space and user-space application

eBPF for Runtime Security Solutions

Toolkits for writing & running arbitrary BPF programs / traces

https://github.com/iovisor/bcc

https://github.com/iovisor/bpftrace

https://github.com/iovisor/kubectl-trace

Multi-use
BPF directly exposed

Platforms built on / using BPF

Targeted Use Cases,
BPF under the covers

eBPF Tech Adoptions

- L3-L4 Load balancing
- Network security
- Traffic optimization
- Profiling

https://code.fb.com/open-source/linux/

Google

- QoS & Traffic optimization
- Network Security
- Profiling
- http://vger.kernel.org/lpc-bpf2018.html#session-1

- Replacing iptables with BPF
- NFV & Load balancing (XDP)
- Profiling & Tracing

https://goo.gl/6JYYJW

NETFLIX

- Performance Troubleshooting
- Tracing & Systems Monitoring
- Networking

http://www.brendangregg.com/blog/2016-03-05/linux-bpf-superpowers.html

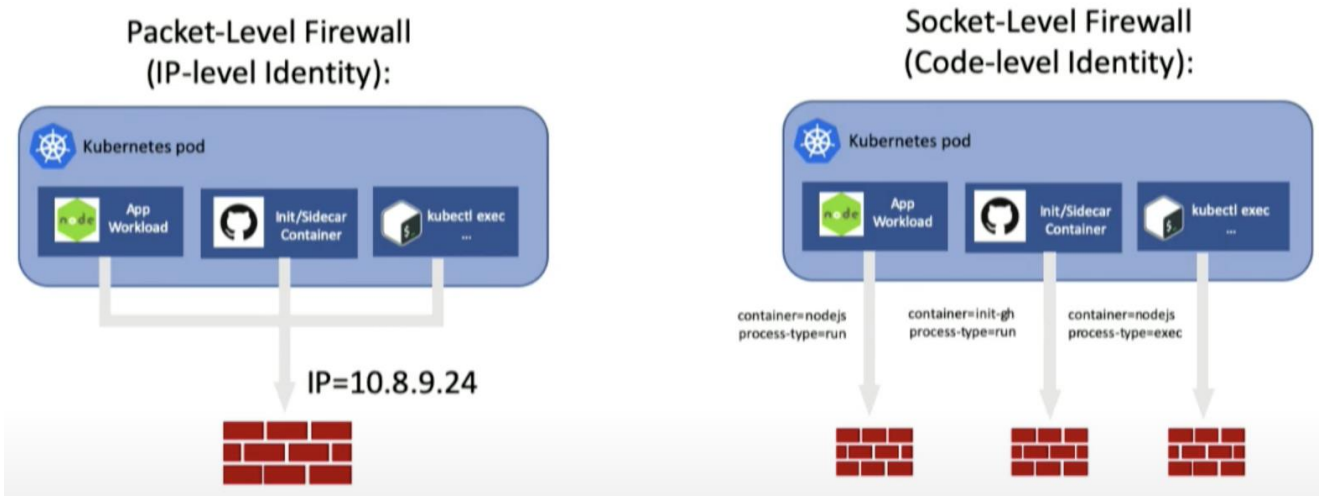
eBPF for Runtime Security: The advantages

- The advantages of eBPF
  - Speed and performance
  - Security
  - Convenience and Programmability
- When eBPF typically works well
  - There's a need for observability using kernel tracing
  - Traditional security monitoring doesn't work

eBPF for Runtime Security

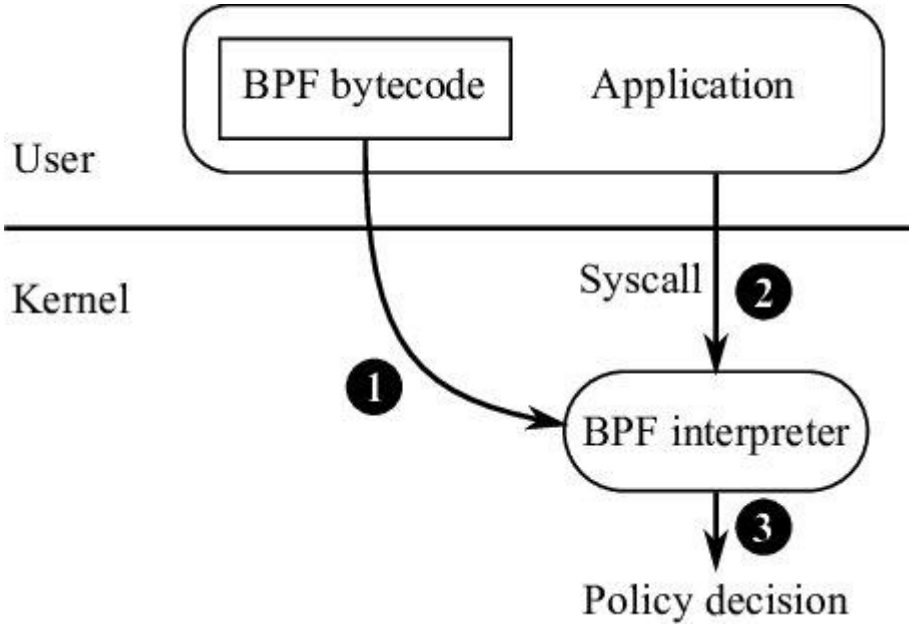
while traditional security monitoring doesn't work

- **Example:** Traditional firewalls are not able to distinguish traffics coming from the same Pod
- eBPF enables **socket-level firewall**



Seccomp: an approach for Privileged Container

- The seccomp help the application to write a filtering function which run before syscalls.
- The filter (written in BPF bytecode) can access only raw syscall arguments



Privileged Container: the limitations

- Require root privilege
  - Allows for unrestricted container management
- Useful for development purposes, but can expose higher risks
  - can start undesirable processes
  - It’s possible to change the user id (UID) or group id (GID)
  - Isolation broken and Privilege escalation

Unprivileged Container

- Container runtime dose not require high privilege for creating and deploying containers ➔ The need of Unprivileged Container technology
- Unprivileged container technology characteristics:
  - Ensure the minimum number of privileges necessary to run a process
  - Can regulate which users in which namespaces are able to execute specific processes
  - Prevent malicious code from gaining permissions in the container host

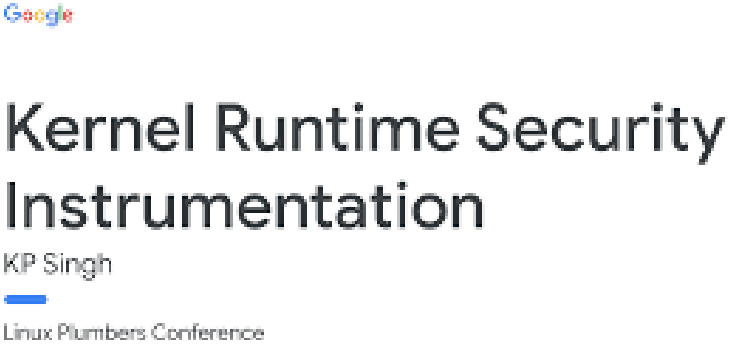
eBPF + LSM: a new approach for making Unprivileged Container Technology

The advantages of eBPF + LSM

Inspecting the container in Kernel semantic	LSM hooks provide powerful mechanism to examine system calls context, including syscall name and its variable
Programmable	By making the Linux kernel programmable, infrastructure software can leverage existing layers, making them more intelligent and feature-rich without continuing to add additional layers of complexity to the system
Checking and Monitoring Syscall from Userspace	The eBPF program is written in Userspace, then attached to LSM hooks and executed in BPF VM in Kernel space
Instrumenting in a safe way	Even eBPF program run in Kernel space, the code runs inside a VM which does not crash or otherwise harm the system

Example projects of eBPF + LSM

- Landlock is geared towards creation of security sandboxes for unprivileged processes.

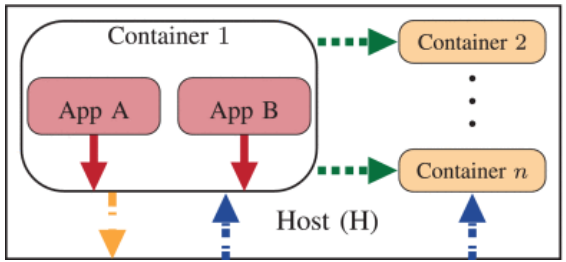


- KRSI gives granular access to security behaviors with an ecosystem of security focused helpers.

4

Conclusion

Summarization

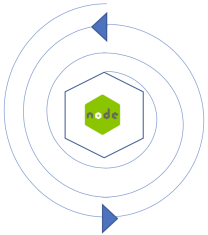


Container security face with many threats

Dockerfile Analysis for Vulnerable Scanning

Static approach for container image scanning

Measure
expected behavior



Monitor
possible deviations

Constrain
to expected behavior

Runtime security = Measure + Monitor + Constrain

\+

Linux Security Module

Unprivileged Security with eBPF + LSM

Research direction

- eBPF + LSM concept are under developing
- It still has a couple of problems:
  - The security issues while sharing data between user-space and kernel-space
  - The performance issues while putting eBPF code inside LSM hooks.

THANKS!

Any questions?