
black hat®
ARSENAL
AUGUST 7-8, 2024
MANDALAY BAY/LAS VEGAS

vArmor : A Sandbox System for Hardening Cloud-Native Containers

Wei Wei, ChangHao Li

About us

- Security researchers and developers at the Endpoint Security team of ByteDance
- Vulnerability and system security research
- Focused on cloud-native security defense construction
- Speakers at Black Hat Europe

Agenda

- Introduction
- Implementation
- Technical Features
- Application Scenarios & Demos
- Next in vArmor

Introduction

The Drive Behind vArmor.

Background

- Container tech revolutionized app development and deployment.
- Kubernetes boosts containerization, devops, and microservices.
- Cloud-Native ≈ Kubernetes-Native
- Introduced new security challenges
  - Weak isolation (shared kernel)
  - Lateral movement hard to defense (overlay & underlay interconnectivity)
- The VM-based Container is not universally applicable and carry some cost.

Linux Container Hardening Techniques

Techniques		Pros	Cons
Least Privilege	Disallow privileged container. Drop capabilities explicitly. Run as non-root users. Set NoNewPrivs. Use read-only root filesystem ...	➢ Easy to configure. ➢ Can increase the difficulty and cost of privilege escalation.	➢ Some mechanisms require application adaptation. ➢ Not applicable to all types of applications (e.g., system and infrastructure-level workloads)
Rootless Container	Puts container runtime and containers in a user ns created by a non-root user.	➢ Can mitigate many vulnerabilities. read/write other users' files, modify the kernel, ARP/DNS spoofing...	➢ Kubernetes doesn't use it by default for compatibility. ➢ Allow non-root users to create user ns increases the kernel's attack surface.
Seccomp	Allows specifying an explicit allowlist or denylist of syscalls	➢ Kubernetes has supported seccomp since v1.3, and it graduated to GA in v1.19. ➢ Can intercept syscalls based on parameters.	➢ Kubernetes doesn't enable it by default for compatibility. ➢ Cannot be modified once it takes effect.
AppArmor	Path-based mandatory access control system.	➢ Kubernetes has supported AppArmor since v1.4. ➢ Fine-grained access control can be implemented.	➢ The default profile of runtimes too loose. ➢ Writing custom profiles requires AppArmor expertise.
SELinux	Label-based mandatory access control system.	➢ Fine-grained access control can be implemented.	➢ The default "container_t" has compatibility issues in some scenarios. ➢ Writing custom profiles requires SELinux expertise.

The Realities

- Workload configurations often fail to meet the baseline profile of the Pod Security Standards.

  —— Your workloads may run live with container escape risks.

- The default profiles are too coarse to defend against some misconfiguration risks, vulnerabilities, and lateral movement.

  —— Some vulnerabilities may have a prolonged repair cycle.

- Building Deny-by-Default profiles for large-scale, fast-changing apps is challenging.

  —— You may lack the expertise, tools and time to craft profiles.

- Managing and applying security policies involve challenges.

  —— You may want to manage and apply profiles in a cloud-native way.

- Some environments do not support AppArmor or SELinux LSM.

  —— You may still want to harden the containers with LSM technology.

The Realities

- Workload configurations often fail to meet the baseline profile of the Pod Security Standards.

  —— Your workloads may run live with container escape risks.

- The default profiles are too coarse to defend against some misconfiguration risks, vulnerabilities, and lateral movement.

  —— Some vulnerabilities may have a prolonged repair cycle.

- Building Deny-by-Default profiles for large-scale, fast-changing apps is challenging.

  —— You may lack the expertise, tools and time to craft profiles.

- Managing and applying security policies involve challenges.

  —— You may want to manage and apply profiles in a cloud-native way.

- Some environments do not support AppArmor or SELinux LSM.

  —— You may still want to harden the containers with LSM technology.

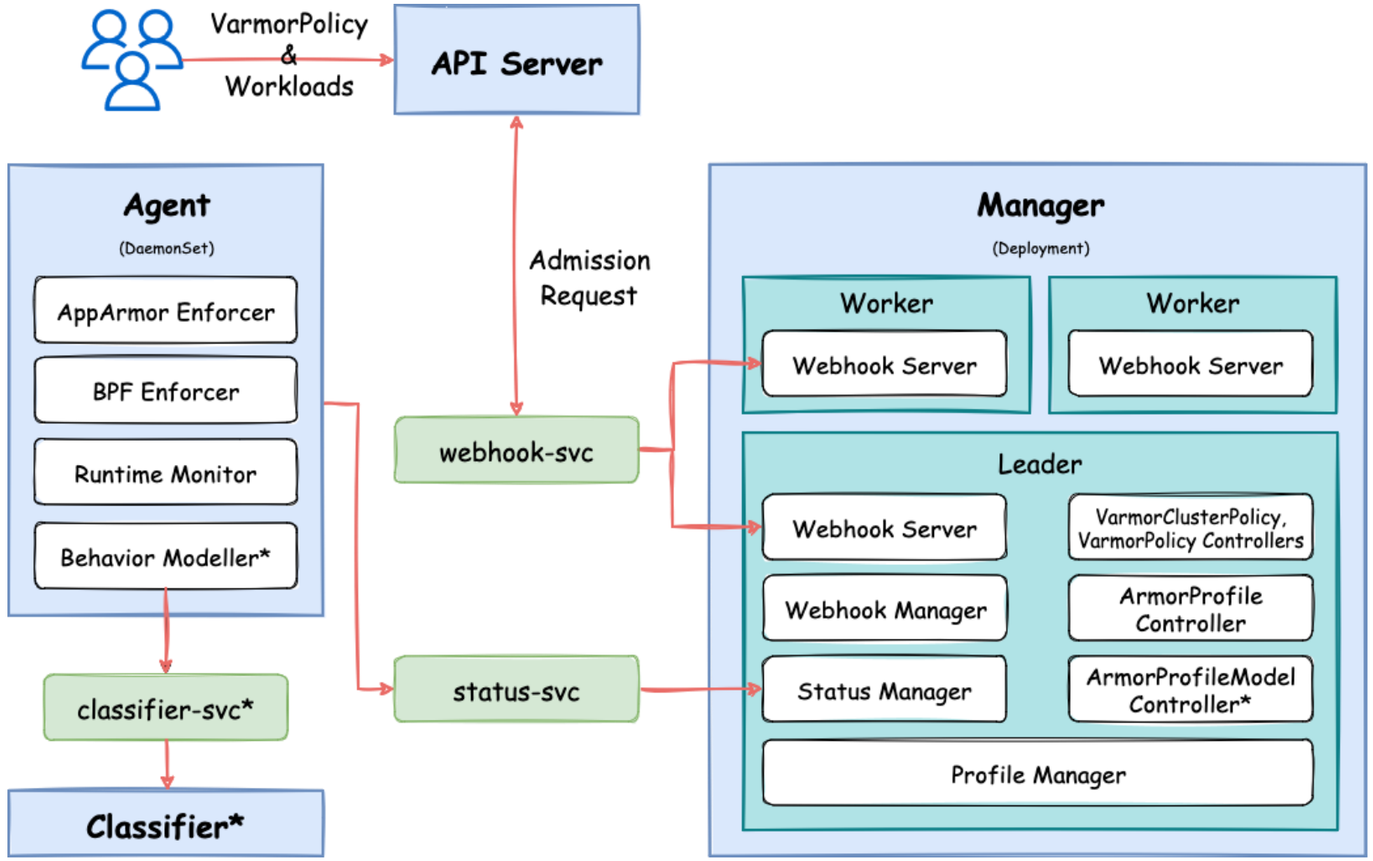
vArmor Born to Solve These Challenges

Implementation

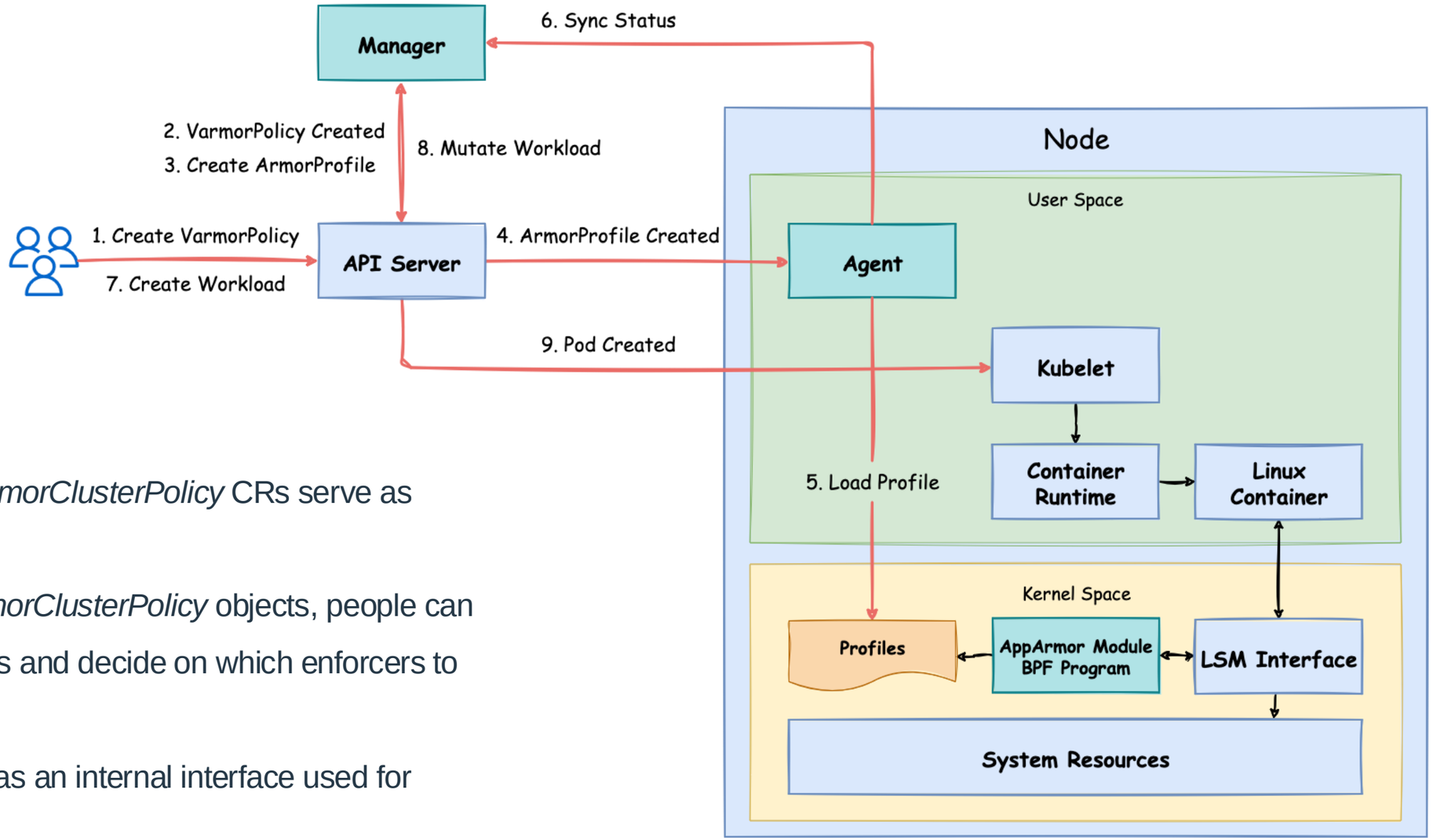
Bridging Container Hardening Gaps in Cloud-Native Environment.

Architecture

- The Custom Resources (CR) are the API interfaces of vArmor.
- Manager acts as an operator to manage the CR objects.
- Agent is responsible for the management of enforcers and profiles.



Principle



- The *VarmorPolicy* and *VarmorClusterPolicy* CRs serve as user interfaces.
- With *VarmorPolicy* or *VarmorClusterPolicy* objects, people can harden particular workloads and decide on which enforcers to use to implement them.
- The *ArmorProfile* CR acts as an internal interface used for profile management.

The AppArmor and Seccomp Enforcers

- The manager creates an *ArmorProfile* object for every *VarmorPolicy* or *VarmorClusterPolicy* object.
- The agents respond to the *ArmorProfile* objects, managing the AppArmor and Seccomp profiles on the node.
- The manager modifies the workloads and configures their *annotations* and *securityContext*.

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: VarmorPolicy
metadata:
  name: demo
spec:
  target:
    kind: Pod
    selector:
      matchLabels:
        app: some
    containers:
    - c1
  policy:
    enforcer: AppArmorSeccomp
    mode: EnhanceProtect
    enhanceProtect:
      hardeningRules:
      - disable-cap-privileged
      attackProtectionRules:
      - rules:
        - disable-chmod-x-bit
```

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: ArmorProfile
metadata:
  name: varmor-default-demo
spec:
  target:
    kind: Pod
    selector:
      matchLabels:
        app: some
    containers:
    - c1
  profile:
    enforcer: AppArmorSeccomp
    content: ${APPARMOR_PROFILE}
    seccompContent: ${SECCOMP_PROFILE}
    ...
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test
  lables:
    app: some
  annotations:
    container.apparmor.security.beta.kubernetes.io/c1: localhost/varmor-default-demo
    container.seccomp.security.beta.varmor.org/c1: localhost/varmor-default-demo
spec:
  containers:
  - name: c1
    securityContext:
      seccompProfile:
        localhostProfile: varmor-default-demo
        type: Localhost
    ...
```

The AppArmor and Seccomp Enforcers

- The manager creates an *ArmorProfile* object for every *VarmorPolicy* or *VarmorClusterPolicy* object.
- The agents respond to the *ArmorProfile* objects, managing the AppArmor and Seccomp profiles on the node.
- The manager modifies the workloads and configures their *annotations* and *securityContext*.

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: VarmorPolicy
metadata:
  name: demo
spec:
  target:
    kind: Pod
    selector:
      matchLabels:
        app: some
    containers:
    - c1
  policy:
    enforcer: AppArmorSeccomp
    mode: EnhanceProtect
    enhanceProtect:
      hardeningRules:
      - disable-cap-privileged
      attackProtectionRules:
      - rules:
        - disable-chmod-x-bit
```

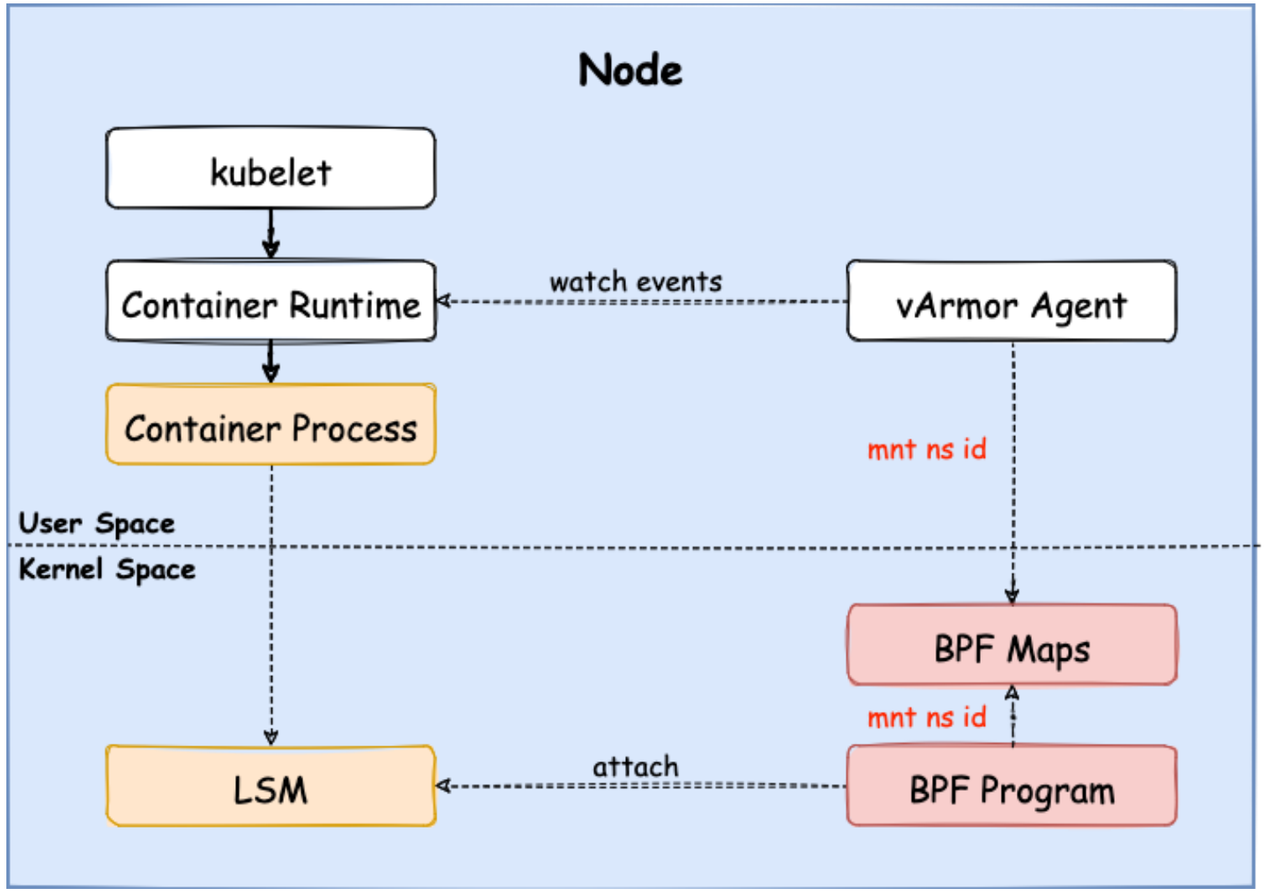
```yaml
apiVersion: crd.varmor.org/v1beta1
kind: ArmorProfile
metadata:
  name: varmor-default-demo
spec:
  target:
    kind: Pod
    selector:
      matchLabels:
        app: some
    containers:
    - c1
  profile:
    enforcer: AppArmorSeccomp
    content: ${APPARMOR_PROFILE}
    seccompContent: ${SECCOMP_PROFILE}
    ...
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test
  lables:
    app: some
  annotations:
    container.apparmor.security.beta.kubernetes.io/c1: localhost/varmor-default-demo
    container.seccomp.security.beta.varmor.org/c1: localhost/varmor-default-demo
spec:
  containers:
  - name: c1
    securityContext:
      seccompProfile:
        localhostProfile: varmor-default-demo
        type: Localhost
    ...
```

How about the BPF enforcer?

The BPF Enforcer

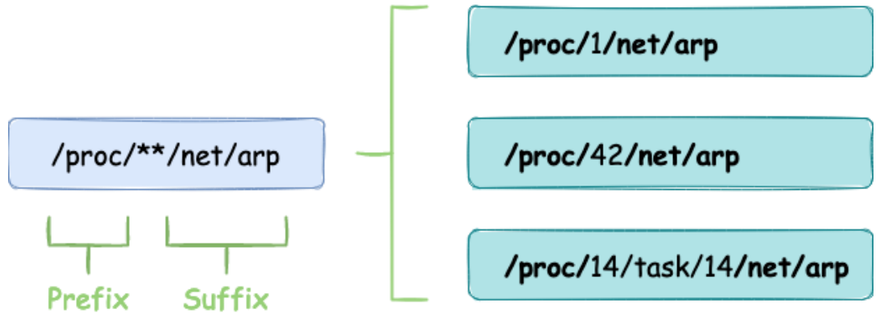
- The Agent monitors container creation and deletion through the Runtime's events mechanism.
- The Agent manages the BPF profiles for containers through the BPF maps.
- The BPF program applies security profile by identifying container processes via their mnt ns id.
- We'll tolerate the TOCTOU risk here, only focusing on long-running services.



The BPF Enforcer

- Implementing access control policy primitives in BPF programs.
- Due to concerns about performance, we will only support the Allow-by-Default security model for now.
- We have utilized a mix of bpf2bpf function calls and tailcalls to implement some policy primitives.

Policy Primitives	Description
Capability	Support access control for capabilities.
File Access	Support access control for read(r), write(w), append(a) permissions on any file, with support for *, ** wildcards.
Process Execution	Support access control for exec(x) permissions on any file, with support for *, ** wildcards.
Network Access	Supports access control for network outbound activities. (CIDR, IP, Port)
Ptrace	Supports access control for ptrace-related permissions. (trace, read, traceby, readby)
Mount	Supports access control for mounting operations. (source, ftype, flags)



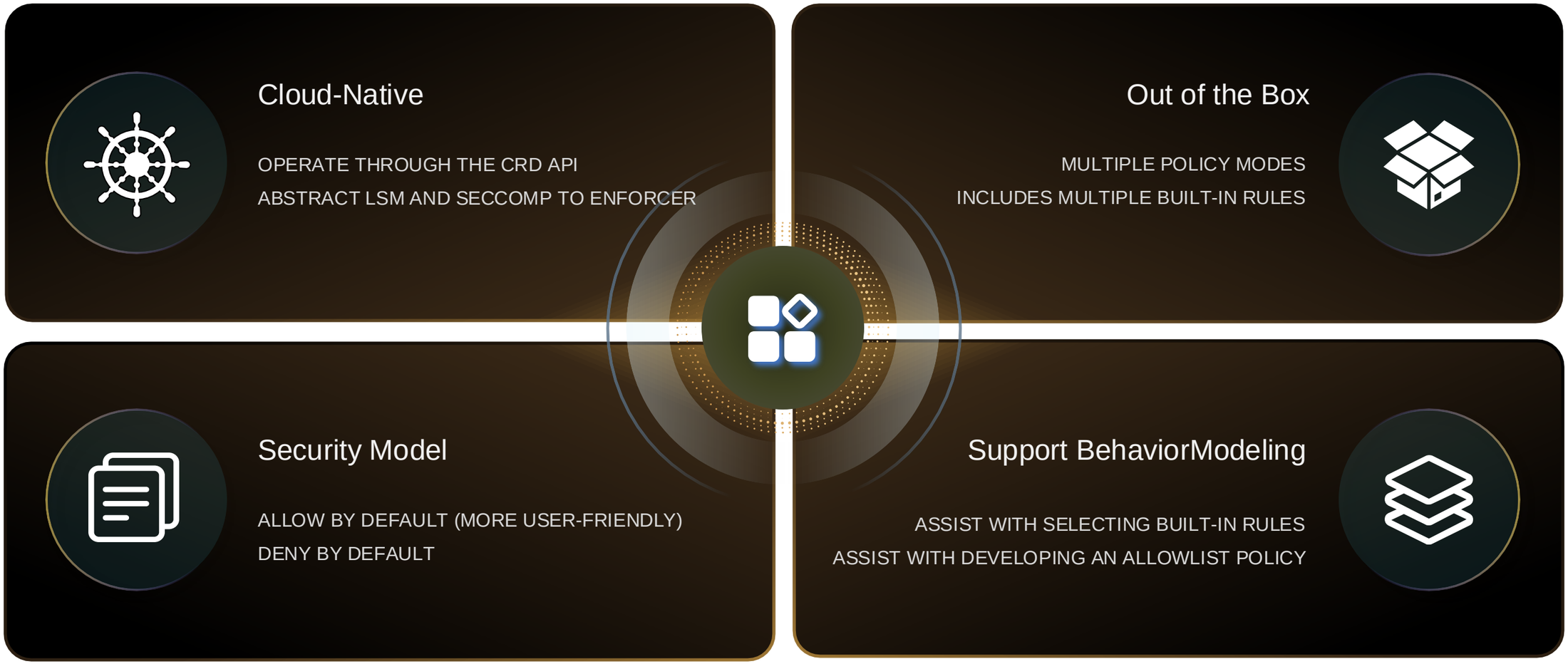
Use prefix and suffix matching to support wildcards.

For more details, please refer to KCon 2023 - Building the Cloud-Native Container Sandbox vArmor from Scratch

Technical Features

Feature Overview and Performance Insights.

Technical Features

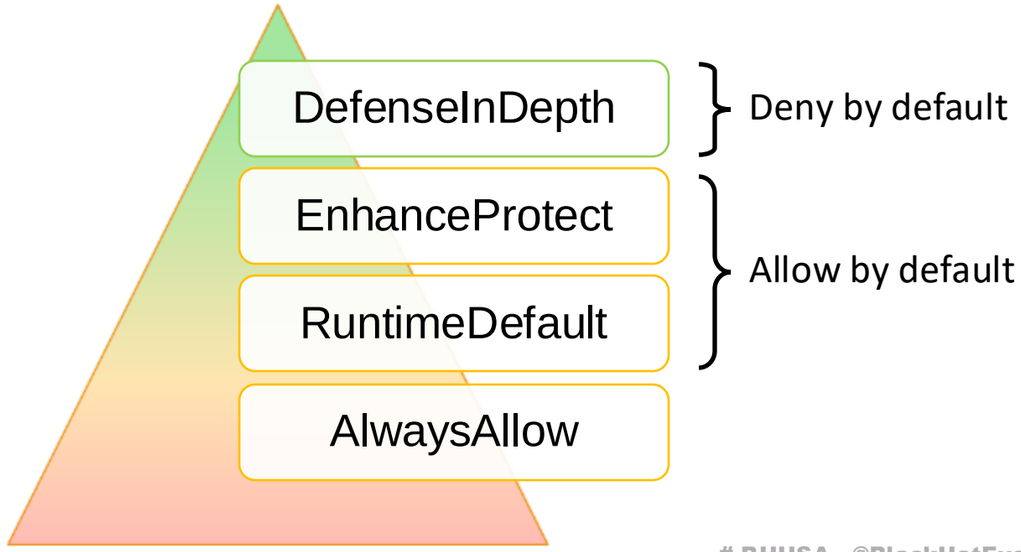


The Policy Modes

- Policies can run in **four modes** and use different **enforcers** to harden workloads.
- Supports updating rules and switching modes without restarting the workloads.
- Policies in EnhanceProtect mode can use **multiple built-in rules** and custom rules.
- *VarmorPolicy* for the namespace-scoped resource.
- *VarmorClusterPolicy* takes effect across the entire cluster.

Policy Mode	AppArmor	BPF	Seccomp
DefenseInDepth	Y	N/A	Y
EnhanceProtect	Y	Y	Y
RuntimeDefault	Y	Y	Y
AlwaysAllow	Y	Y	Y

Note: The workloads must be restarted for changes to the Seccomp Profile to take effect when you are using the Seccomp enforcer.



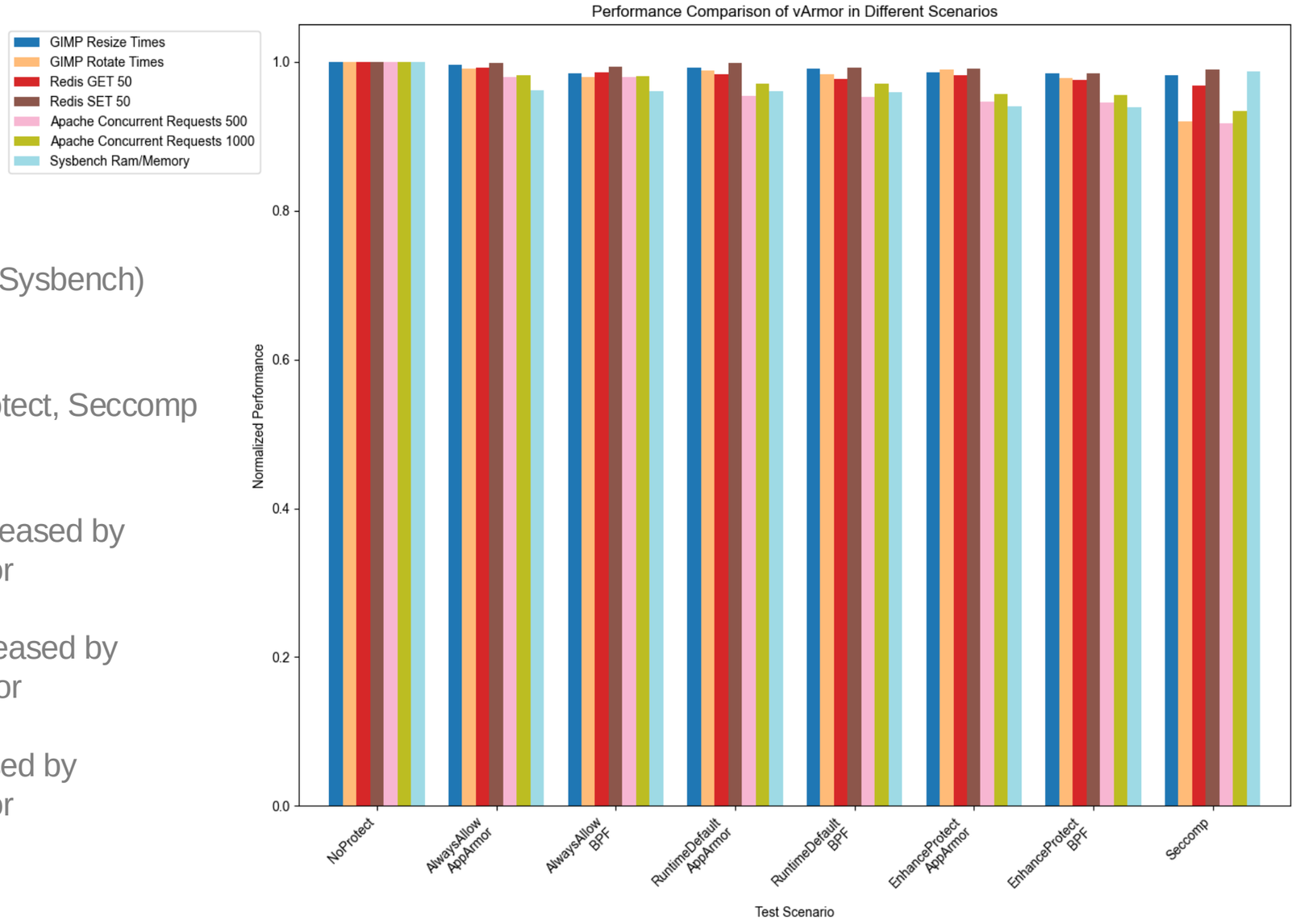
The Built-in Rules

- We have implemented a multiple built-in rules based on the policy primitives of enforcers.
- We also support custom interfaces, allowing users to define custom rules based on syntax.
- vArmor will help users generate the final profiles.

Category		Description
Hardening	Rules to reduce the attack surface of system.	➢ Block common escape vectors for privileged containers. ➢ Disable capabilities. ➢ Block certain kernel vulnerability exploitation vectors.
Attack Protection	Rules against penetration tactics in the container environment.	➢ Mitigate container information leakage. ➢ Prohibit execution of sensitive actions.
Vulnerability Mitigation	Rules for mitigating specific vulnerabilities.	➢ cgroups-lxcfs-escape mitigation ➢ runc override mitigation ➢ dirty-pipe mitigation

https://github.com/bytedance/vArmor/blob/main/docs/built_in_rules.md

The Performance



Tools used

- phoronix-test-suite (Redis, Apache, GIMP, Sysbench)

Strategies

- AlwaysAllow, RuntimeDefault, EnhanceProtect, Seccomp

Summary

- EnhanceProtect: Performance of BPF decreased by approximately 1.2% compared to AppArmor
- RuntimeDefault: Performance of BPF decreased by approximately 0.6% compared to AppArmor
- AlwaysAllow: Performance of BPF decreased by approximately 0.1% compared to AppArmor

https://github.com/bytedance/vArmor/blob/main/docs/performance_specification.md

Application Scenarios & Demos

Showcasing vArmor in Action.

Reduce the Attack Surface of Kernel

- Take the the attack surface introduced by *User Namespace* as an example.
- Allowing non-root users to create user namespaces has increased the kernel attack surface.

```
test@ncie3ms95qc6ihoeubf80:~$ id
uid=1000(test) gid=1000(test) groups=1000(test)
test@ncie3ms95qc6ihoeubf80:~$ cat /proc/self/status | grep CapEff
CapEff: 0000000000000000
test@ncie3ms95qc6ihoeubf80:~$ readlink /proc/self/ns/user
user:[4026531837]
test@ncie3ms95qc6ihoeubf80:~$ unshare --user --map-root-user
root@ncie3ms95qc6ihoeubf80:~# readlink /proc/self/ns/user
user:[4026532894]
root@ncie3ms95qc6ihoeubf80:~# cat /proc/self/status | grep CapEff
CapEff: 000001fffffffff
root@ncie3ms95qc6ihoeubf80:~# unshare --net
root@ncie3ms95qc6ihoeubf80:~# ip link add name dummy0 type dummy
root@ncie3ms95qc6ihoeubf80:~# ip link set dummy0 up
root@ncie3ms95qc6ihoeubf80:~# ip addr add 192.168.1.1/24 dev dummy0
root@ncie3ms95qc6ihoeubf80:~# ip addr show dummy0
2: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 46:6b:82:0e:c3:cd brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 scope global dummy0
       valid_lft forever preferred_lft forever
    inet6 fe80::446b:82ff:fe0e:c3cd/64 scope link
       valid_lft forever preferred_lft forever
```

Reduce the Attack Surface of Kernel

- Take the the attack surface introduced by *User Namespace* as an example.
- Allowing non-root users to create user namespaces has increased the kernel attack surface.
- Vulnerabilities caused by unprivileged user namespaces aren't decreasing anytime soon.

  CVE-2021-22555, CVE-2022-0185, CVE-2022-25636, CVE-2022-2588, CVE-2023-32233...
- Current defense techniques

  user.max_user_namespaces=0 (may conflict with normal business operations)

  kernel.unprivileged_userns_clone=0 (non-mainstream)

  disable unshare syscall with SeccompDefault profile (disabled by default for compatibility sake)

  LSM hook: userns_create (Linux v6.1+)

vArmor Built-in Rules	Description	Supported Enforcer
Prohibit abusing user namespaces disallow-abuse-user-ns	Disallowing container processes from abusing CAP_SYS_ADMIN privileges via user namespaces.	AppArmor, BPF
Prohibit creating user namespace disallow-create-user-ns	Disallowing container processes from creating new user namespaces.	Seccomp

Reduce the Attack Surface of Kernel

- Take the the attack surface introduced by *User Namespace* as an example.

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: VarmorPolicy
metadata:
  name: cve-2021-22555
  namespace: demo
spec:
 target:
    kind: Deployment
    selector:
      matchLabels:
        app: cve-2021-22555
  policy:
    enforcer: AppArmor
    mode: EnhanceProtect
    enhanceProtect:
      hardeningRules:
      - disallow-abuse-user-ns
```

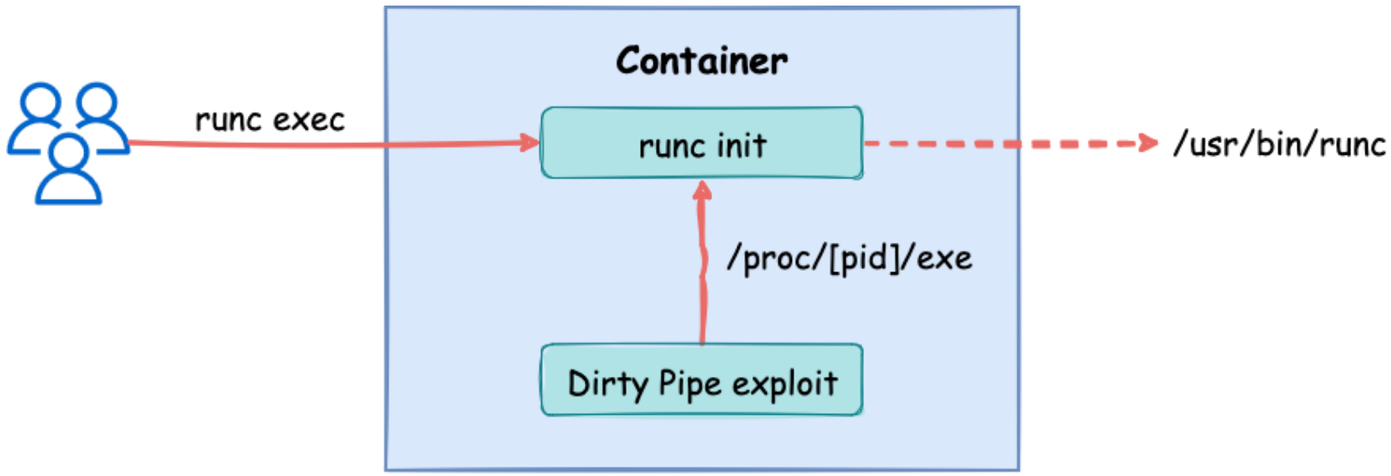
https://github.com/bytedance/vArmor/tree/main/test/demo/vulnerability-mitigation/CVE-2021-22555

Mitigate Specific Vulnerabilities

- Take *Dirty Pipe* (CVE-2022-0847) as an example.

The Dirty Pipe is a vulnerability in the Linux kernel since 5.8 which allows overwriting data in arbitrary read-only files. This leads to privilege escalation because unprivileged processes can inject code into root processes.

It is similar to CVE-2016-5195 “Dirty Cow” but is easier to exploit.



Mitigate Specific Vulnerabilities

- Take *Dirty Pipe* (CVE-2022-0847) as an example.

  Disallow calling SPLICE(2) syscall.

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: VarmorPolicy
metadata:
  name: cve-2022-0847
  namespace: demo
spec:
 target:
    kind: Deployment
    selector:
      matchLabels:
        app: cve-2022-0847
  policy:
    enforcer: Seccomp
    mode: EnhanceProtect
    enhanceProtect:
      vulMitigationRules:
      - dirty-pipe-mitigation
  updateExistingWorkloads: true
```

https://github.com/bytedance/vArmor/tree/main/test/demo/vulnerability-mitigation/CVE-2022-0847-dirty-pipe

Harden Containers that have Privileges

- Take a container that has CAP_SYS_ADMIN as an example.
- CAP_SYS_ADMIN is more secure than ‘privileged: true’
- CAP_SYS_ADMIN == Container Escape

```
mount -t proc tmpproc /tmp/proc
mount --bind /proc /tmp/proc
mount --bind /proc/sys /tmp/proc
mount --move /proc/keys /proc/kcore
umount /proc/sys && echo "xxx" > /proc/sys/kernel/core_pattern
mount -t cgroup -o devices -o rw tmpcgroup /tmp/cgroup
mount --bind /sys/fs/cgroup/devices /tmp/cgroup/devices
mount -t securityfs securityfs /tmp/security
...
```

Harden Containers that have Privileges

- Take a container that has CAP_SYS_ADMIN as an example.

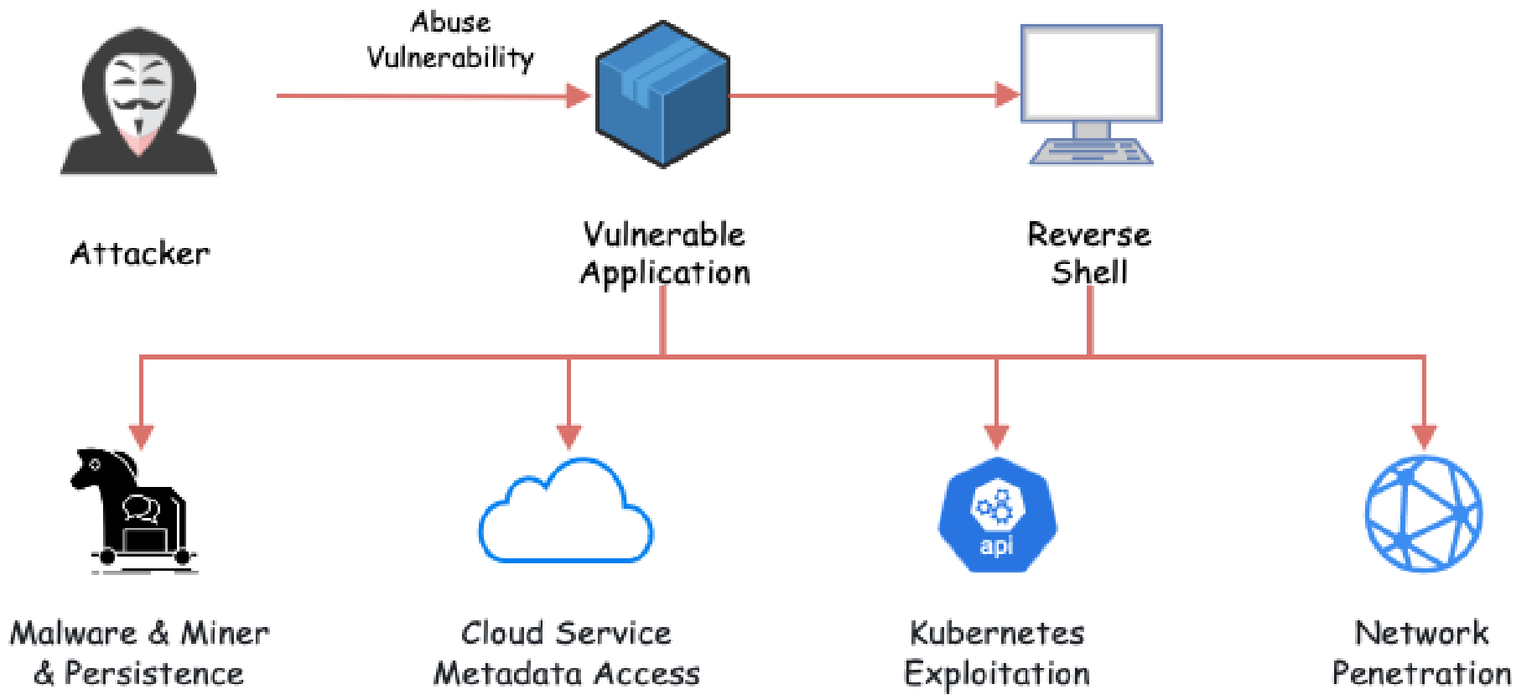
  Prohibit mount, remount, and umount on cgroupfs and procfs...

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: VarmorClusterPolicy
metadata:
  name: sys-admin-app-policy
spec:
 target:
    kind: Deployment
    selector:
      matchLabels:
        app: sys-admin-app
 policy:
    enforcer: BPF
    mode: EnhanceProtect
    enhanceProtect:
      hardeningRules:
      - disallow-write-core-pattern
      - disallow-mount-securityfs
      - disallow-mount-procfs
      - disallow-mount-cgroupfs
      privileged: true
 updateExistingWorkloads: true
```

https://github.com/bytedance/vArmor/tree/main/test/demo/vulnerability-mitigation/SYS_ADMIN

Defending Against Penetration Tactics

- Take the vulnerability of the *Confluence* (CVE-2022-26134) as an example.

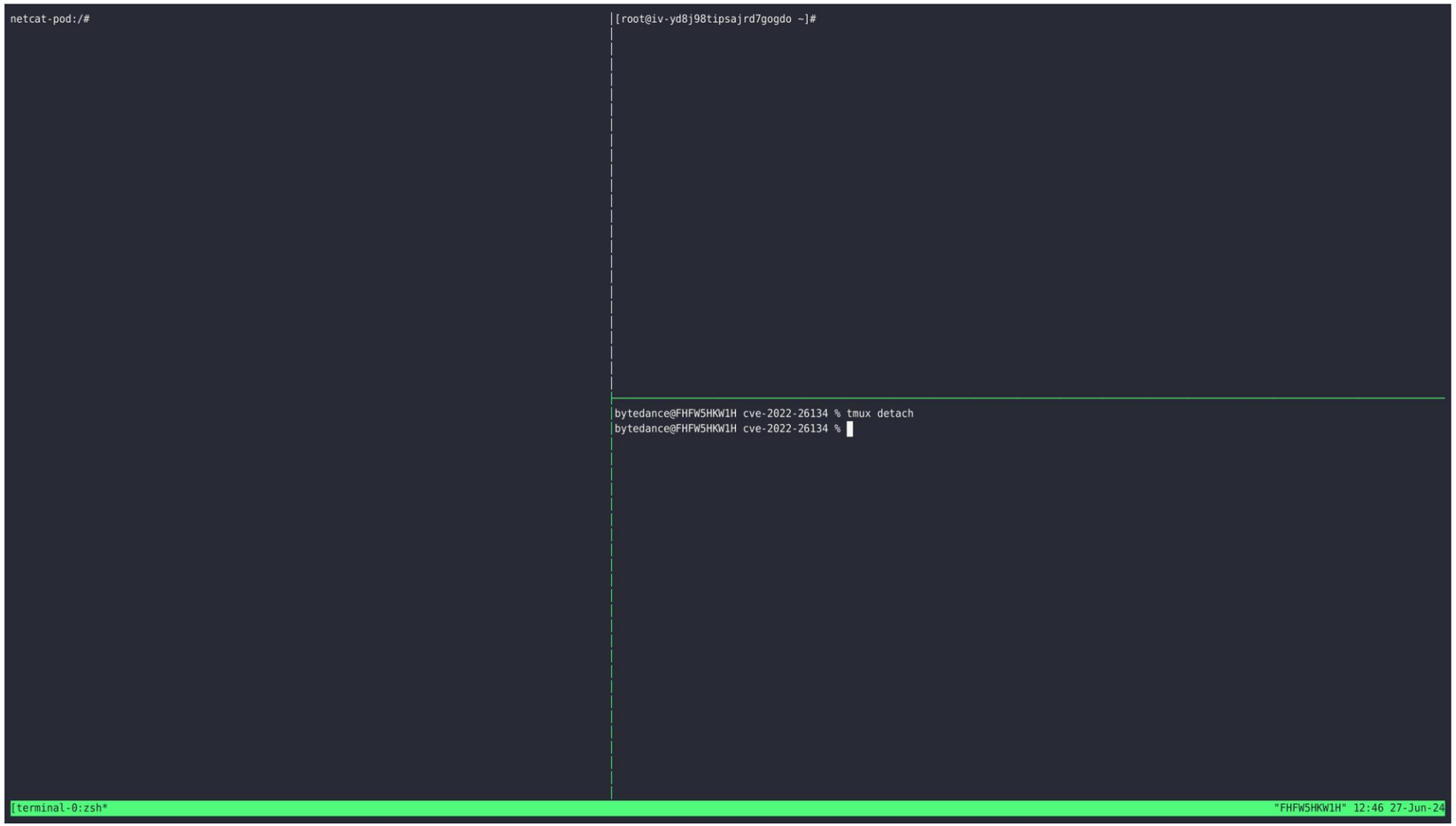


Cloud Native Scenario: Typical Attack Path (e.g., CVE-2022-26134)

Defending Against Penetration Tactics

- Take the vulnerability of the *Confluence* (CVE-2022-26134) as an example.

```yaml
apiVersion: crd.varmor.org/v1beta1
kind: VarmorPolicy
metadata:
  name: bpf-enhance
spec:
  target:
    kind: Deployment
    selector:
      matchLabels:
        app: protect
  policy:
    enforcer: BPFSeccomp
    mode: EnhanceProtect
    enhanceProtect:
      hardeningRules:
      - disable-cap-privileged
      - disallow-access-procfs-root
      attackProtectionRules:
      - rules:
        - disable-write-etc
        - mitigate-sa-leak
        - disable-curl
        - disallow-metadata-service
        - disable-chmod
        - disable-shell
        - mitigate-host-ip-leak
```



https://github.com/bytedance/vArmor/tree/main/test/demo/vulnerability-mitigation/CVE-2022-26134

Next in vArmor

- Introduce new features
  * behavior modeling and violations audit for BPF Enforcer
  * more policy primitives for BPF Enforcer
  * more built-in rules
  * ...
- Test coverage
- Observability
- Compatibility
- Performance

Welcome to the community!

vArmor

GitHub

https://github.com/bytedance/vArmor

Thanks

Appendix: Enforcer Prerequisites

Enforcer	Requirements	Recommendations
AppArmor	• Linux Kernel 4.15 and above • The AppArmor LSM is enabled	• GKE with Container-Optimized OS • AKS with Ubuntu 22.04 LTS • VKE with veLinux • Debian 10 and above • Ubuntu 18.04.0 LTS and above • veLinux etc.
BPF	• Linux Kernel 5.10 and above (x86_64) • containerd v1.6.0 and above • The BPF LSM is enabled	• EKS with Amazon Linux 2 • GKE with Container-Optimized OS • VKE with veLinux (with 5.10 kernel) • AKS with Ubuntu 22.04 LTS * • ACK with Alibaba Cloud Linux 3 * • OpenSUSE 15.4 * • Debian 11 * • Fedora 37 • veLinux with 5.10 kernel etc. * *Manual enabling of BPF LSM is required*
Seccomp	• Kubernetes v1.19 and above	All Linux distributions

Appendix: Installation and Configuration

```
helm pull oci://elkeid-ap-southeast-1.cr.volces.com/varmor/varmor --version 0.5.11

helm install varmor varmor-0.5.11.tgz  \
   --namespace varmor \
   --create-namespace \
   --set image.registry="elkeid-ap-southeast-1.cr.volces.com" \
   --set bpfLsmEnforcer.enabled=true
```

Helm Options	Description
--set appArmorLsmEnforcer.enabled=false	Default: enabled. The AppArmor enforcer can be disabled if the system does not support AppArmor LSM.
--set bpfLsmEnforcer.enabled=true	Default: disabled. The BPF enforcer can be enabled if the system supports BPF LSM.
--set behaviorModeling.enabled=true	Default: disabled. Experimental feature. Currently, only the AppArmor/Seccomp enforcer supports the BehaviorModeling mode.
--set "manager.args={--webhookMatchLabel=KEY=VALUE}"	The default value is: sandbox.varmor.org/enable=true. vArmor will only enable sandbox protection for Workloads that contain this label. You can disable this feature by using --set 'manager.args={--webhookMatchLabel=}'.

https://github.com/bytedance/vArmor/blob/main/docs/usage_instructions.md

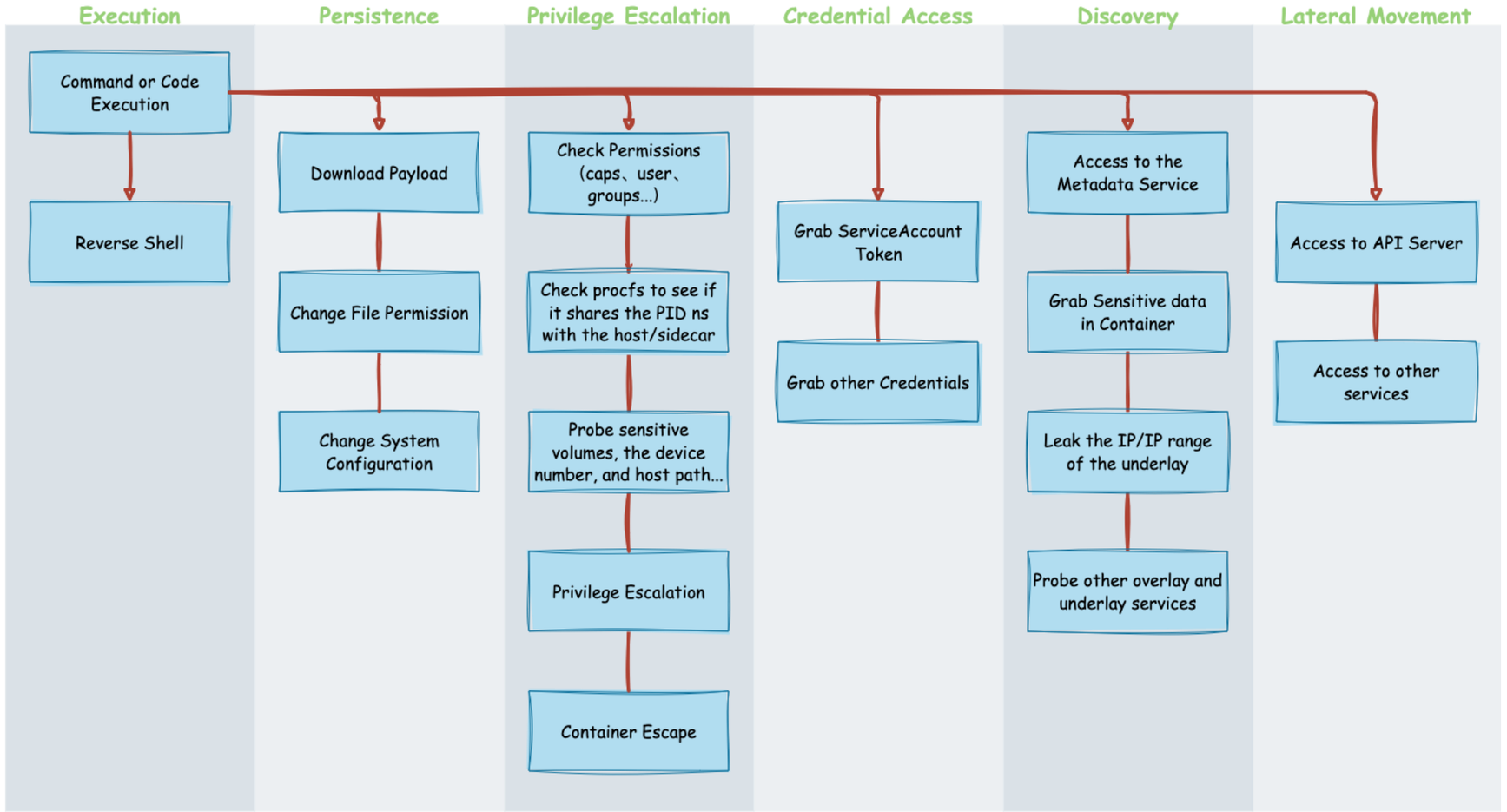
Appendix: Target Identify

Field	Description
kind *string*	Kind is used to specify the type of workloads for the protection targets. Available values: *Deployment*, *StatefulSet*, *DaemonSet*, *Pod*
name *string*	Optional. Name is used to specify a specific workload name.
selector *LabelSelector*	Optional. *LabelSelector* is used to match workloads that meet the specified conditions. Note: the type of workloads is determined by the *KIND* field.
containers *string array*	Optional. Containers are used to specify the names of the protected containers. If it is empty, the security policy will be applied for all containers within the workload (excluding *initContainers* and *ephemeralContainers*).

```
spec:
  target:
    kind: Deployment
      selector:
        matchExpressions:
        - key: environment
          operator: In
          values: [qa, prod]
        matchLabels:
          app: demo-1
      containers: [c0]
```

Appendix: Penetration Tactics in Containers

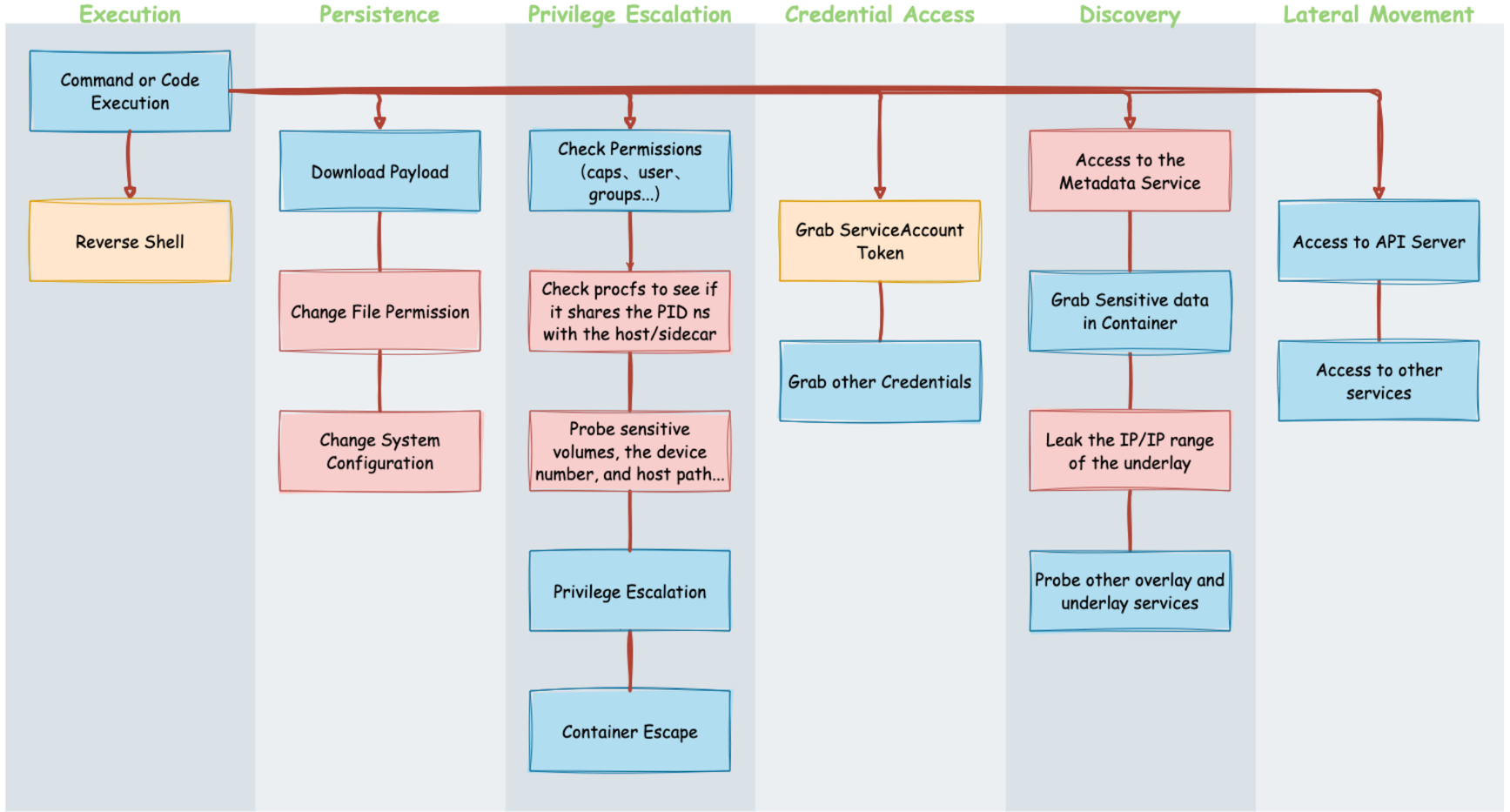
- The normal behavior of the application within the container, denoted as A.
- Intrusion behavior by the attacker within the container, denoted as B.
- Behavior that can be restricted by the sandbox, denoted as $C = B - (A \cap B)$



https://github.com/bytedance/vArmor/blob/main/docs/built_in_rules.md

Appendix: Penetration Tactics in Containers

- AppArmor & BPF enforcer
  - mitigate-sa-leak
  - mitigate-disk-device-number-leak
  - mitigate-overlayfs-leak
  - mitigate-host-ip-leak
  - disable-write-etc
  - disable-busybox
  - disable-shell
  - disable-wget
  - disable-curl
  - disable-chmod
  - disable-su-sudo
- BPF enforcer
  - disallow-metadata-service
- Seccomp enforcer
  - disable-chmod-x-bit
  - disable-chmod-s-bit



https://github.com/bytedance/vArmor/blob/main/docs/built_in_rules.md

Appendix: Custom Rules

- Users can write custom “deny rules” in EnhanceProtect mode.
- Refer to the syntax for AppArmor.
- Refer to the bpfrawrules for BPF.
- Refer to the LinuxSyscall for Seccomp.

```
enforcer: AppArmorBPFSeccomp
mode: EnhanceProtect
enhanceProtect:
 # The custom AppArmor rules
 appArmorRawRules:
 - "deny /etc/shadow r,”

 # The custom BPF rules
 bpfRawRules:
  network:
   egresses:
   - ipBlock: 2001:db8::/32
     port: 443
  files:
   pattern: /bin/*
   permissions: [“w”, “a”]

 # The custom Seccomp rules
 syscallRawRules:
 # disallow chmod +x, chmod 111, chmod 001, chmod 010
 - names:
  - fchmodat
  action: SCMP_ACT_ERRNO
  args:
  - index: 2
   value: 0x40    # S_IXUSR
   valueTwo: 0x40
   op: SCMP_CMP_MASKED_EQ
  - index: 2
   value: 0x8     # S_IXGRP
   valueTwo: 0x8
   op: SCMP_CMP_MASKED_EQ
  - index: 2
   value: 1       # S_IXOTH
   valueTwo: 1
   op: SCMP_CMP_MASKED_EQ
```

Appendix: Behavior Modeling (Experimental)

- Use BPF and Audit to perform behavior modeling on multiple workloads.
- The behavior data will be stored in the corresponding ArmorProfileModel object.
- Only support AppArmor & Seccomp for now.
- Use cases
  - Assist with selecting built-in rules (See Policy Advisor).
  - Assist with developing an allowlist profile.
  - Guide the workload to minimize the securityContext.

```
policy:
  enforcer: AppArmorSeccomp
  # Capture the behavior data of target workloads,
  # Collect and aggregate the behavior data of target workloads.
  mode: BehaviorModeling
  modelingOptions:
    # The duration in minutes to modeling
    duration: 3
```

```
# Generate a ".spec.policy" template in the EnhanceProtect mode
# with built-in rules and behavior data.
policy-advisor.py [-h] [-f FEATURES] [-c CAPABILITIES]
                  [-m BEHAVIOR_MODEL] [-d]
                  enforcers
```

Use Case 1

```
policy:
  enforcer: AppArmorSeccomp
  # Use the allowlist profile generated from the behavior data
  # to enforce access control.
  mode: DefenseInDepth
```

Use Case 2