Пых . конф'25

Сергей Канибор

Luntry

Путеводитель по безопасности контейнеров для PHP-разработчиков

ispring latech TRIBE STUDIO DATS TEAM Eventico RUVENTS

whoami

- R&D / Container Security в Luntry
- Специализируюсь на безопасности контейнеров и Kubernetes
- Автор CVE и BDU
- Багхантер
- Редактор Telegram канала k8s(in)security
- Спикер крупнейших ИТ и ИБ конференций

Пых . конф'25

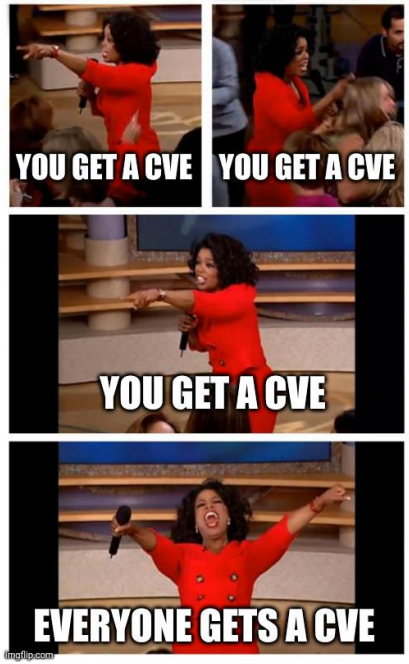
План

- Какие бывают угрозы в контейнерах
- Distroless образы
- Харденинг
- Пример
- Выводы

Какие бывают угрозы в контейнерах

CVE повсюду

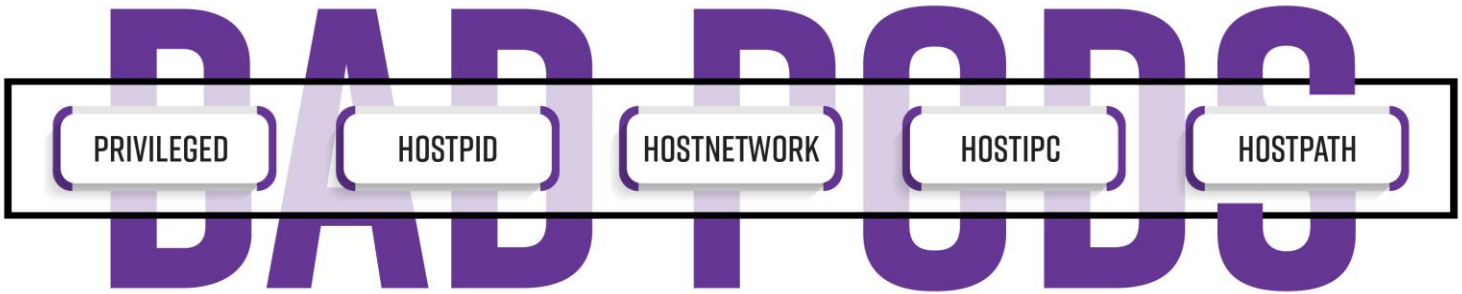
- Уязвимости в базовых образах
- Уязвимости в зависимостях приложения
- Уязвимости в пакетах хостовых ОС
- Уязвимости в ядре
- Уязвимости в компонентах K8s
- Еще какие-то уязвимости...



Пых . конф'25

Bad Pods

- Некорректно сконфигурированные манифесты для нагрузок
- Их эксплуатация может привести к повышению привилегий



Bad Pods – Privileged

- Privileged == ALL capabilities
- Монтирование хостовой системы
- Побег из контейнера через cgroups
- Побег через различные user mode helpers

```
apiVersion: v1
kind: Pod
metadata:
  name: priv-exec-pod
  labels:
    app: pentest
spec:
  containers:
  - name: priv-pod
  image: ubuntu
  securityContext:
    privileged: true
  command: [ "/bin/sh", "-c", "--" ]
  args: [ "while true; do sleep 30; done;" ]
```

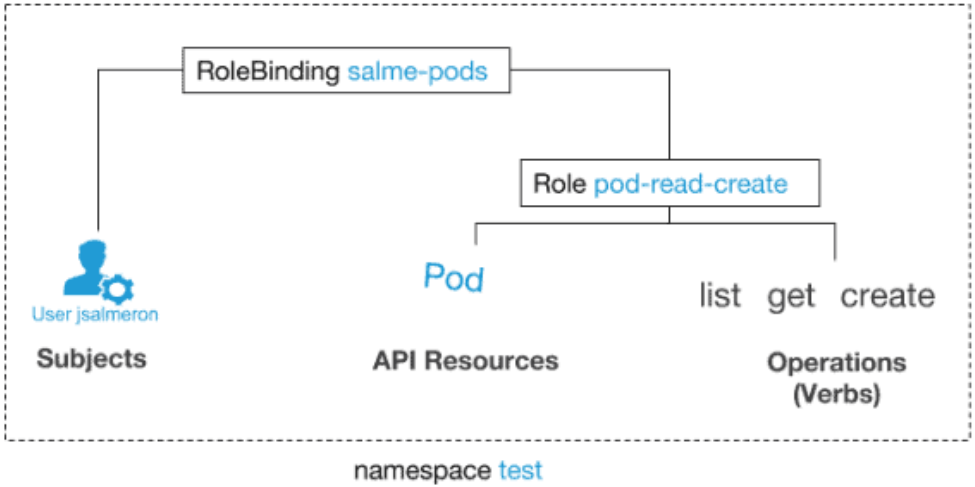
Bad Pods – hostPID

- Видно все процессы хоста
- Видно все файловые дескрипторы
- Можно собирать пароли, ключи, токены и т.д с чужих контейнеров
- Или просто убить чужой процесс (контейнер)

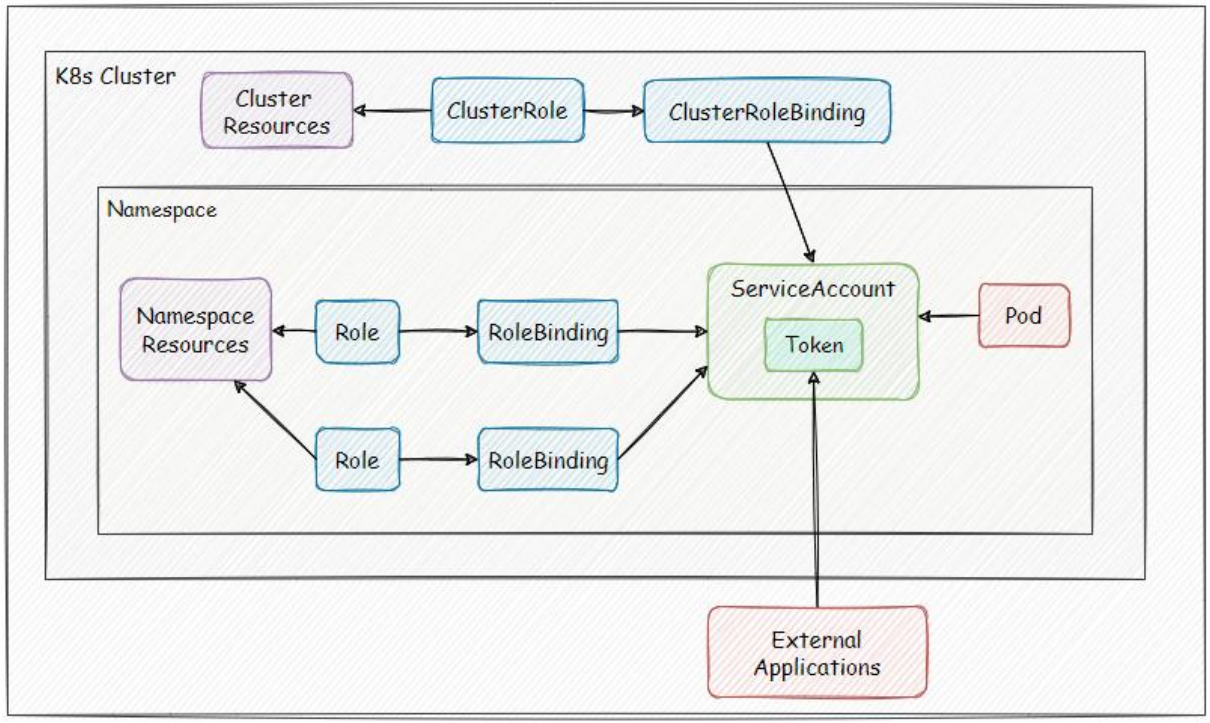
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: priv-exec-pod
  labels:
    app: pentest
spec:
  hostPID: true
  containers:
  - name: priv-pod
  image: ubuntu
  command: [ "/bin/sh", "-c", "--" ]
  args: [ "while true; do sleep 30; done;" ]
```

RBAC

- Ролевая модель в K8s
- Иногда бывают излишние права
- Иногда дают разрешения,
- не зная их значений
- Как следствие – повышение привилегий и захват кластера

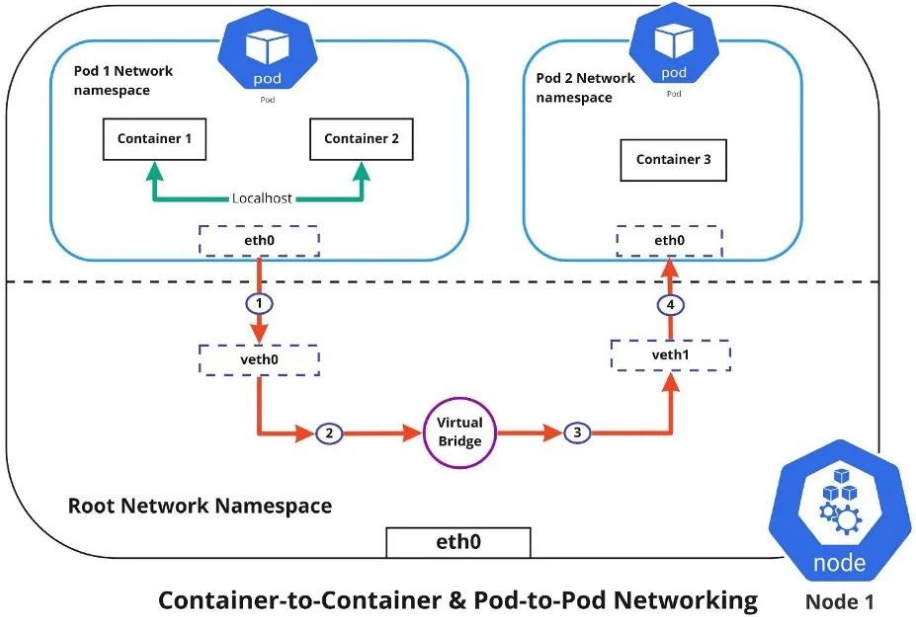


Service Account



Kubernetes Network

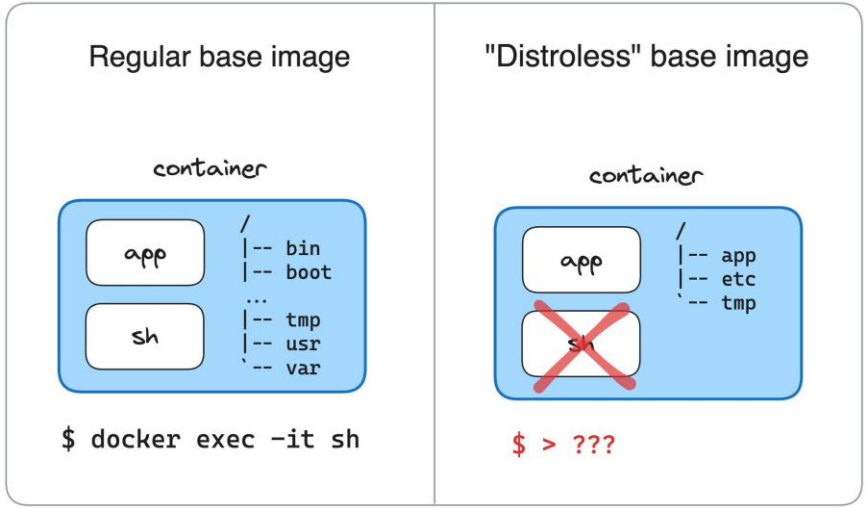
- По умолчанию сеть плоская
- Каждый может ходить к каждому
- hostNetwork дает доступ к сети хоста



Distroless образы

Distroless images

- Дословно переводится как «без дистрибутива»
- Из образа выпиливаются все ненужные бинари и библиотеки
- В итоге получаем образ в котором есть условно «один бинарь»



Пых . конф'25

Distroless примеры

- Google distroless
- Chainguard
- Самопис

Пример

Я не знаю PHP 😄

Я не знаю PHP 😄

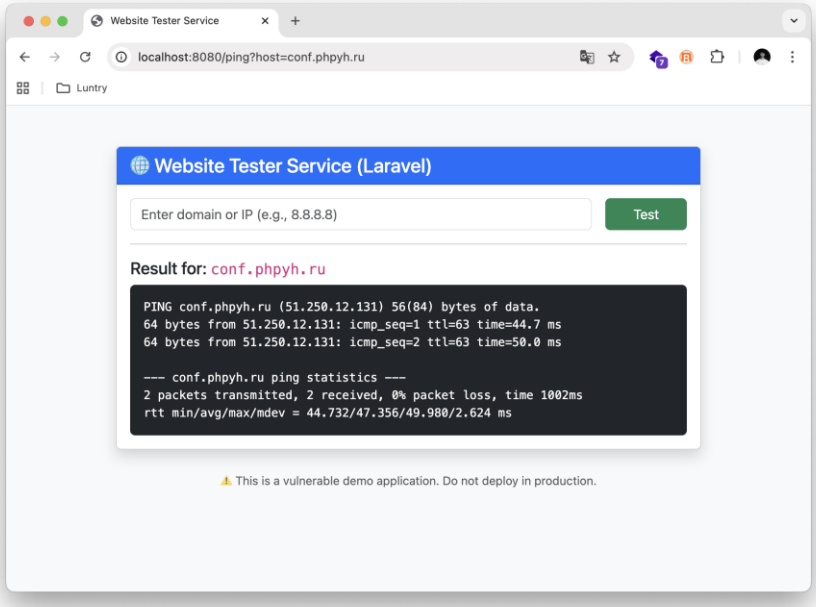
Напиши небольшое веб приложение на PHP, которое будет уязвимо к Command injection. Приложение похоже на "Websites Tester Service", но с уязвимостью

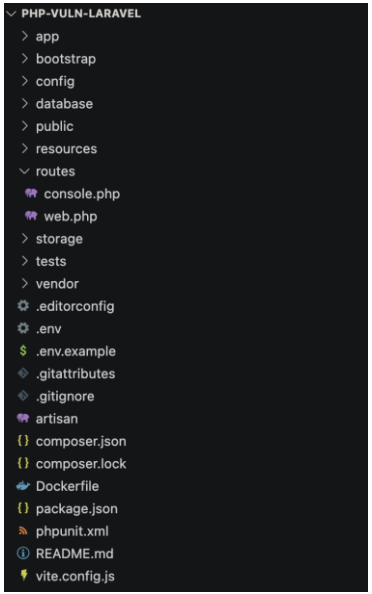
Я хочу запустить это приложение в докер контейнере. Напиши для него dockerfile

Напиши это приложение с использованием фреймворка Laravel

Я не знаю PHP 😁

- В итоге я получил вот это





Пых . конф'25

PingController

- Где тут уязвимость?

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PingController extends Controller
{
    public function index()
    {
        return view('ping');
    }

    public function ping(Request $request)
    {
        $host = $request->query('host');
        $output = '';

        if ($host) {
            $command = "ping -c 2 " . $host;
            $output = shell_exec($command);
        }

        return view('ping', compact('output', 'host'));
    }
```

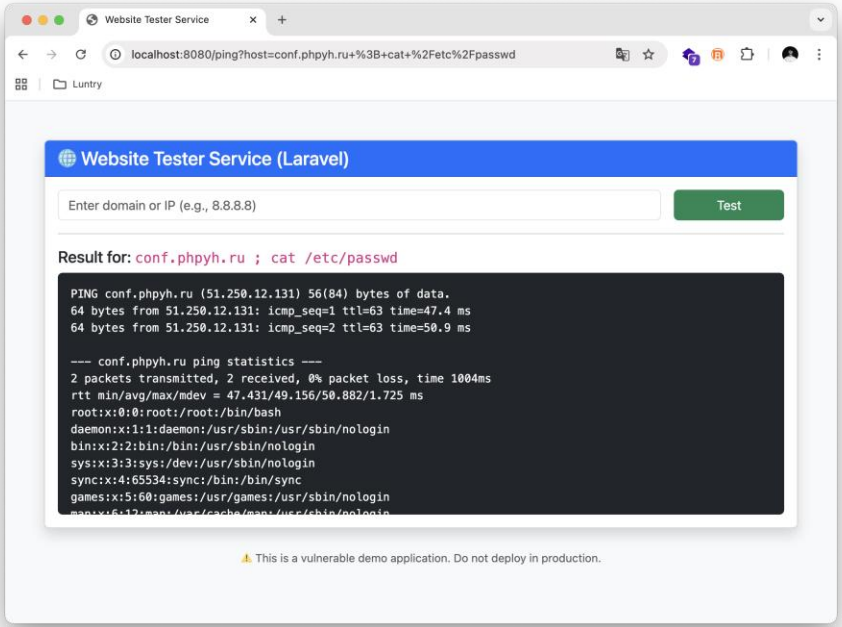
PingController

- Где тут уязвимость?
- Нет фильтрации ввода пользователя
- Весь инпут просто передается в shell_exec

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PingController extends Controller
{
    public function index()
    {
        return view('ping');
    }

    public function ping(Request $request)
    {
        $host = $request->query('host');
        $output = '';

        if ($host) {
            $command = "ping -c 2 " . $host;
            $output = shell_exec($command);
        }

        return view('ping', compact('output', 'host'));
    }
```

PingController

- Где тут уязвимость?
- Нет фильтрации ввода пользователя
- Весь инпут просто передается в shell_exec
- В итоге можно выполнять произвольный код



Пых . конф'25

Обычный базовый образ

- Приложение запущено

  на базе php:8.3-fpm

```
FROM php:8.3-fpm

RUN apt-get update && apt-get install -y \
unzip zip git curl libzip-dev libonig-dev \
&& docker-php-ext-install pdo pdo_mysql zip
```

Начинаем приседать

- Используем distroless от chainguard
- Для начала нужен composer, чтобы подтянуть зависимости
- Подготавливаем проект к запуск

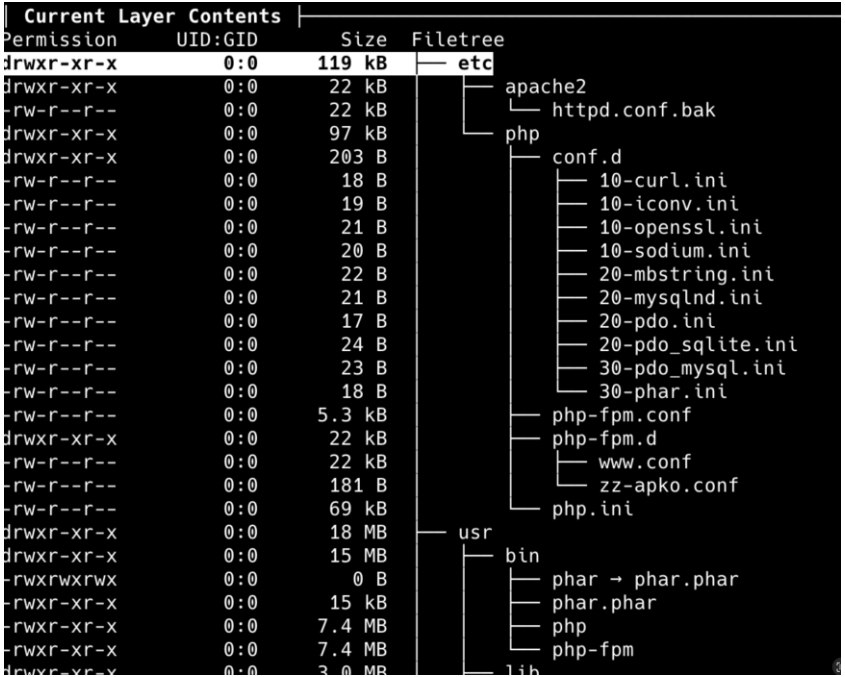
```
FROM cgr.dev/chainguard/php:latest-dev AS build

WORKDIR /app

RUN composer install --no-interaction --prefer-dist \
&& cp .env.example .env || true \
&& php artisan key:generate || true
```

Пых . конф'25

Докидываем рантайм

- Используем distroless от chainguard для рантайма
- В нем ничего нет кроме бинаря php

```
FROM cgr.dev/chainguard/php:latest-fpm AS runtime

WORKDIR /app
COPY ./nping /usr/bin/nping

EXPOSE 9000

CMD ["php-fpm"]
```



Пых . конф'25

Докидываем nginx

- Добавим nginx для сервинга

```
FROM cgr.dev/chainguard/nginx AS nginx

WORKDIR /app

EXPOSE 8080
```

Пых . конф'25

Сборка готова

- Получаем такую multistage сборку

```
FROM cgr.dev/chainguard/php:latest-dev AS build
WORKDIR /app
RUN composer install --no-interaction --prefer-dist \
&& cp .env.example .env || true \
&& php artisan key:generate || true
FROM cgr.dev/chainguard/php:latest-fpm AS runtime
WORKDIR /app
COPY ./nping /usr/bin/nping
EXPOSE 9000
CMD ["php-fpm"]
FROM cgr.dev/chainguard/nginx AS nginx
WORKDIR /app
EXPOSE 8080
```

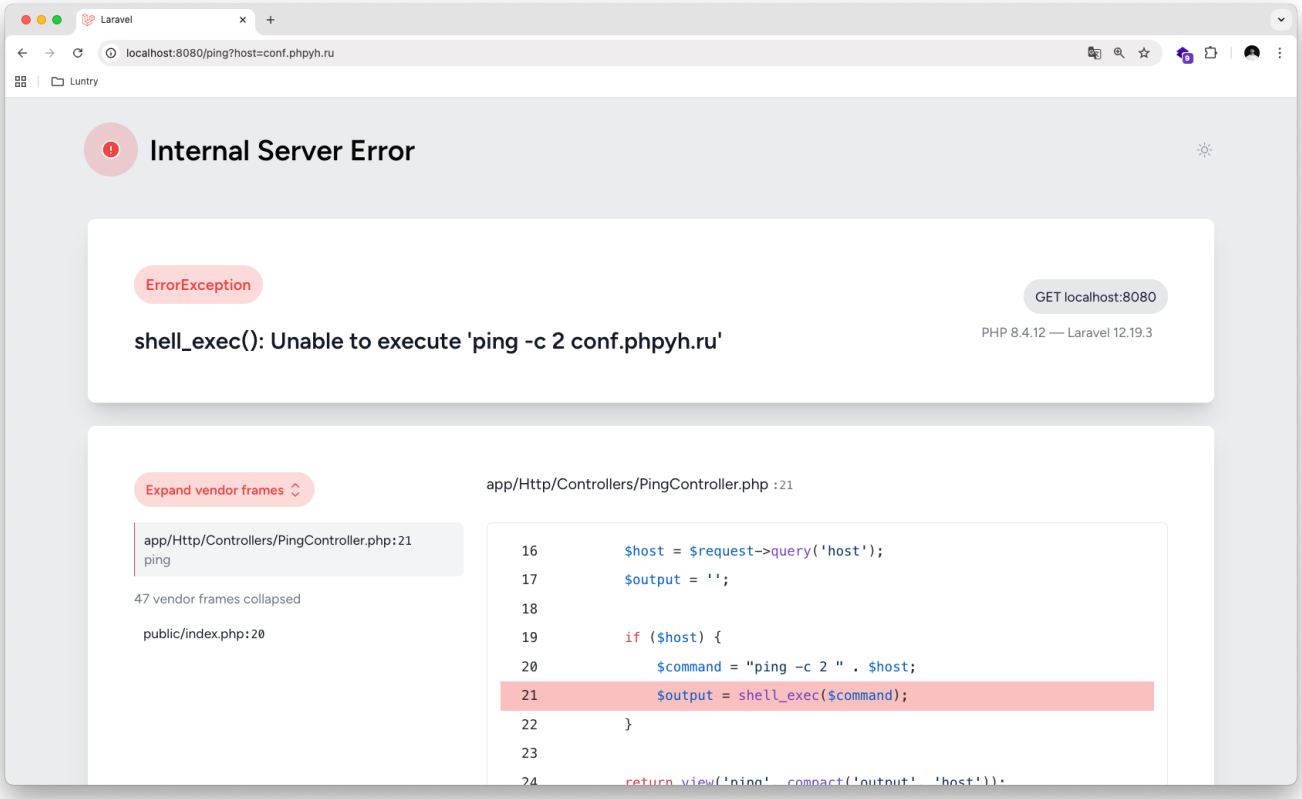
Используем compose

- Запускаем через docker compose

```yaml
services:
  app:
    build:
      context: .
      target: runtime
    restart: unless-stopped
    working_dir: /app
    volumes:
      - ./app:/app

  nginx:
    build:
      context: .
      target: nginx
    restart: unless-stopped
    ports:
      - 8080:8080
    volumes:
      - ./app:/app
      - ./nginx.conf:/etc/nginx/nginx.conf
```

Нельзя так просто взять..

Потому что вот так



Пых . конф'25

Как было

- Ping вызывался через shell_exec
- То есть дальше он передавался /bin/sh
- Но теперь /bin/sh нет

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PingController extends Controller
{
    public function index()
    {
        return view('ping');
    }

    public function ping(Request $request)
    {
        $host = $request->query('host');
        $output = '';

        if ($host) {
            $command = "ping -c 2 " . $host;
            $output = shell_exec($command);
        }

        return view('ping', compact('output', 'host'));
    }
```

Как стало

- Заменяем shell_exec на proc_open
- Теперь вызывается конкретный бинарь
- В образе нет ничего лишнего
- Потому что по другому никак =)

```php
private function runNping(array $args): string
{
    $cmd = array_merge(['/usr/bin/nping'], $args);

    $descriptorspec = [
        0 => ['pipe', 'r'], // stdin
        1 => ['pipe', 'w'], // stdout
        2 => ['pipe', 'w'], // stderr
    ];

    $process = proc_open($cmd, $descriptorspec, $pipes);

    if (!is_resource($process)) {
        throw new RuntimeException("Try again start nping");
    }
```

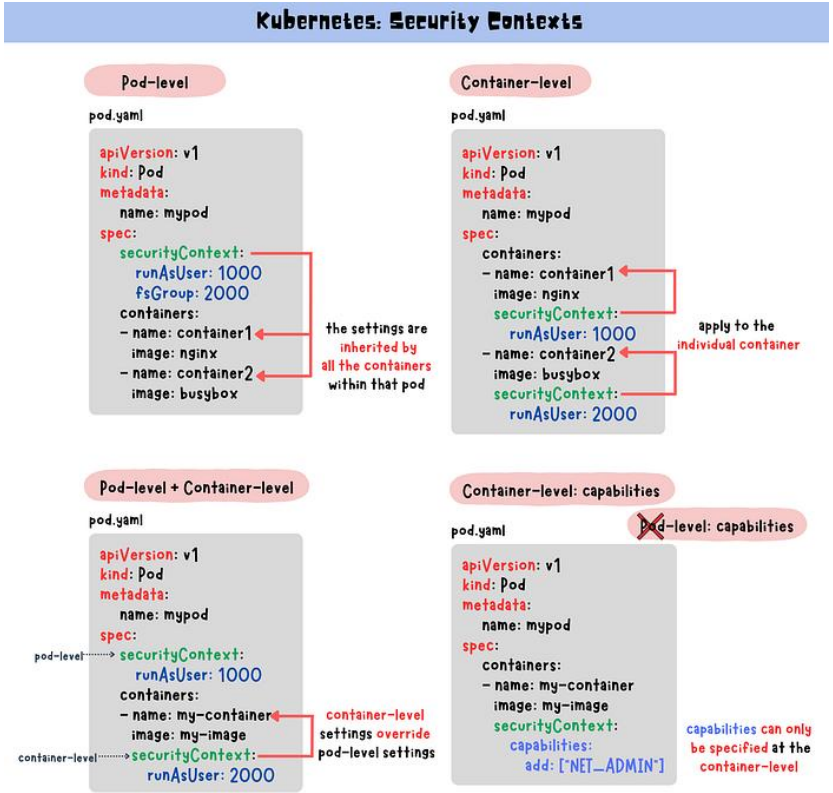
Харденинг

А что со стороны K8s?

```yaml
  annotations:
    container.apparmor.security.beta.kubernetes.io/app: runtime/default
spec:
  hostNetwork: false
  hostPID: false
  hostIPC: false
  automountServiceAccountToken: false
  securityContext:
    seccompProfile:
      type: RuntimeDefault
    runAsNonRoot: true
    runAsUser: 10001
    runAsGroup: 10001
    fsGroup: 20001
  shareProcessNamespace: false
```

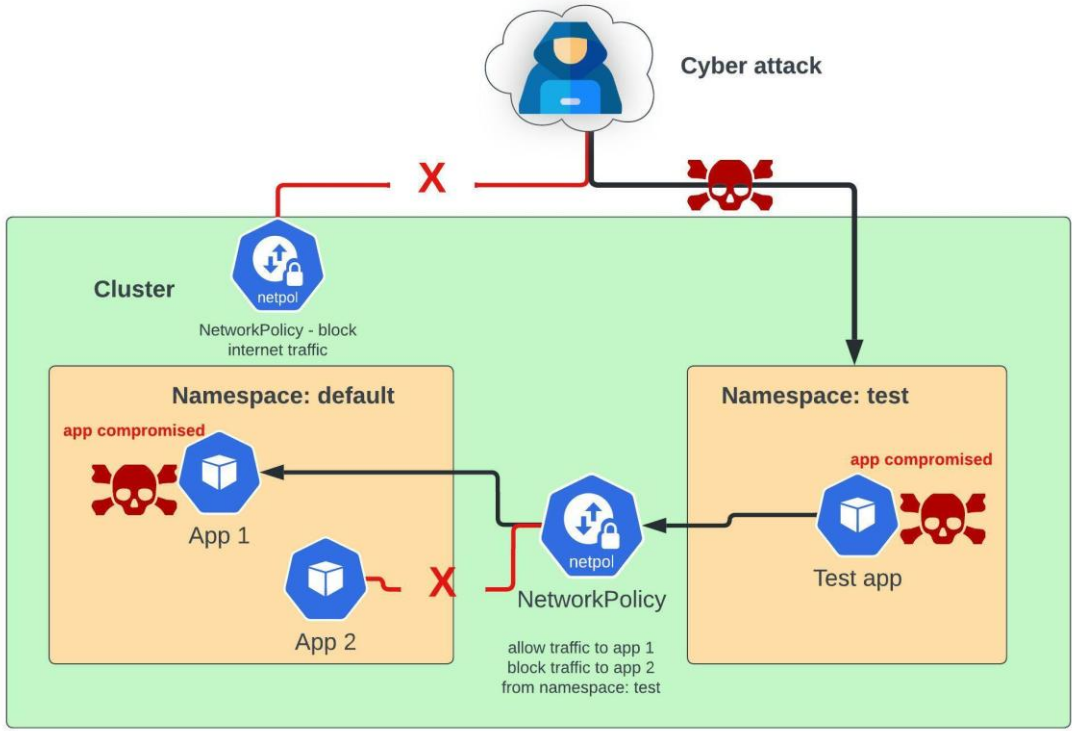
```yaml
containers:
  - name: app
    image: gcr.io/distroless/static@sha256:pyhconf
    imagePullPolicy: Always
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop: ["ALL"]
      seLinuxOptions:
        type: "container_t"
    volumeMounts:
      - name: tmp
        mountPath: /tmp
        readOnly: false
      - name: config
        mountPath: /app/config
        readOnly: true
```

Kubernetes securityContext



Пых . конф'25

Network Policy



Network Policy – пример

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

Выводы

- Можно писать безопасный код, но всегда есть человеческий фактор
- Distroless – хороший способ сделать образы
- для запуска PHP приложений легкими и безопасными
- Kubernetes – даёт широкие возможности для безопасной конфигурации нагрузок

Спасибо за внимание!

Пожалуйста, оставьте отзыв

Сергей Канибор

Luntry

Путеводитель по безопасности контейнеров для PHP-разработчиков

@rObinak

sk@luntry.ru

habr @rObinak

Пых . конф'25