ПРОФЕССИОНАЛИТЕТ, 2025, № 1 (005)

***УДК** 004.492.3*

БЕЗОПАСНАЯ CI-ПРОВЕРКА С ПОМОЩЬЮ FALCO ДЛЯ ОБНАРУЖЕНИЯ АТАК В KUBERNETES ПЕРЕД РАЗВЁРТЫВАНИЕМ

Т. Р. Абдуллин [1]

[1]*ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация*

Аннотация: *В данной статье рассматривается подход к повышению безопасности предрелизной CI-проверки за счёт интеграции системы обнаружения угроз Falco в Kubernetes-кластер. Актуальность работы обусловлена ростом числа атак на контейнеризированные веб-приложения и необходимостью выявления подозрительной активности не только на уровне системных вызовов, но и на уровне прикладного взаимодействия. Основная проблема заключается в том, что стандартная конфигурация Falco ориентирована преимущественно на мониторинг действий внутри контейнеров и на хосте, поэтому её возможностей недостаточно для обнаружения веб-атак, таких как SQL-инъекции, XSS, path traversal и попытки доступа к конфиденциальным файлам. Для решения данной задачи предлагается расширить функциональность Falco с помощью кастомного плагина анализа логов Nginx и специализированных правил детектирования. В ходе исследования разработан экспериментальный стенд на базе Kubernetes, включающий уязвимое веб-приложение OWASP Juice Shop, веб-сервер Nginx и систему мониторинга Falco. Дополнительно реализован автоматизированная CI-проверка на основе GitHub Actions, обеспечивающий развёртывание тестовой среды и проверку корректности работы настроенных правил. Полученные результаты показывают, что предложенный подход позволяет обнаруживать различные виды веб-атак в реальном времени и может использоваться как элемент повышения защищённости DevOps-процессов при эксплуатации контейнеризированных приложений.*

Ключевые слова: *Kubernetes, CI, Falco, веб-атака, контейнеризация, мониторинг безопасности, Nginx.*

Введение

В современном мире с каждым днём растёт количество интернет-ресурсов, которые зачастую представляют собой веб-приложения. Вместе с тем, увеличивается и количество инцидентов в сфере информационной безопасности, потому что при недостаточной защищённости веб-приложения представляют собой точку входа для злоумышленников. По мере противостояния кибератакам появляются новые решения в сфере безопасности, что затрудняет процесс проникновения злоумышленников в систему, однако полностью обезопасить интернет-ресурс невозможно.

В связи с этим стоит вопрос не только активной защиты информации, но и вопрос обнаружения угроз в реальном времени. Одним из таких средств обнаружения является инструмент для мониторинга Falco. Он используется для отслеживания и анализа попыток совершения системных вызовов, изменений в файловой системе и сетевой активности, с целью выявления подозрительных или вредоносных действий.

Одним из современных средств развёртывания веб-инфраструктуры являются технологии контейнеризации и оркестрации, поэтому, в данной статье, Falco будет рассмотрен на примере его внедрения в Kubernetes.

Исходя из вышесказанного, целью данной работы является исследование и практическое применение методов обнаружения угроз веб-приложений в реальном времени в Kubernetes с использованием Falco.

Оркестрация и Kubernetes

Как было отмечено ранее, одним из способов развёртывания веб-приложений является развёртывание с использованием технологий контейнеризации. Такой подход обеспечивает лёгкость, переносимость и ресурсоэффективность. Однако по мере роста количества сервисов, составляющих современное веб-приложение, растёт и количество контейнеров, которыми необходимо управлять, а значит повышается и сложность управления. Возникают сложные задачи: обеспечение отказоустойчивости, автоматическое масштабирование под нагрузкой, управление сетевым взаимодействием между компонентами и централизованное обновление версий.

Именно здесь на первый план выходят технологии оркестрации контейнеров. Они позволяют автоматизировать весь жизненный цикл контейнеризованных приложений, абстрагируя инфраструктуру в единую управляемую среду. Одним из средств оркестрации является Kubernetes. Kubernetes – это портативная расширяемая платформа с открытым исходным кодом для управления контейнеризованными рабочими нагрузками и сервисами, которая облегчает как декларативную настройку, так и автоматизацию.

Основная ценность Kubernetes заключается в декларативном подходе к управлению инфраструктурой. Вместо того чтобы вручную запускать контейнеры на конкретных серверах, администратор описывает желаемое состояние системы, а Kubernetes автоматически и непрерывно приводит реальное состояние кластера к описанному, устраняя возникающие расхождения.

Когда разработчик работает с Kubernetes, он имеет дело с кластером. Kubernetes-кластер – набор машин (физических или виртуальных), называемых узлами (или нодами, node), которые запускают контейнеризированные приложения. Кластер имеет как минимум один рабочий узел. На узлах кластера находятся:

1. Control Plane (Управляющий слой) – набор компонентов, управляющих состоянием кластера и контролирующих его работу. Данные компоненты отвечает за основные операции кластера: обработка запросов к кластеру, хранение всех данных кластера, распределение подов на worker ноды, мониторинг состояния кластера и выполнение действий по исправлению этого состояния.

2. Worker Nodes (Рабочие узлы) – узлы, на которых выполняются поды. Под – это наименьшая развёртываемая единица в Kubernetes, это абстракция, которая может содержать чаще один и реже несколько контейнеров, разделяющих общие аппаратные ресурсы узла.

Основы пайплайнов CI/CD

Одним из ключевых этапов жизненного цикла приложения является его проверка и доставка до целевой среды с последующим развертыванием. В современных DevOps-практиках эти процессы объединяются в концепцию непрерывной интеграции и непрерывного развертывания (CI/CD), реализуемую через автоматизированные пайплайны. CI и CD – это аббревиатуры от Continuous Integration (непрерывная интеграция) и Continuous Deployment (непрерывное развертывание). Continuous Integration – процесс интегрирования своего кода в общий репозиторий компании. Данный процесс сопровождается автоматическими тестами, которые проверяют работу кода и следят за тем, чтобы новый функционал не нарушал уже существующий. Continuous Deployment – процесс автоматического применения изменений и их развёртывания в среде для тестирования, или в среде, доступной пользователю.

Предрелизная CI-проверка в свою очередь – это поток работ, включающих CI и CD, он описывает последовательность шагов, выполняемых в автоматическом или полуавтоматическом режиме с момента внесения изменений в репозиторий до их внедрения в рабочую систему [1]. В контексте контейнеризованных приложений и Kubernetes пайплайн CI/CD обретает особую значимость. Он становится тем самым механизмом, который преобразует исходный код разработчика в работающие поды внутри кластера.

Инструмент мониторинга Falco

В предыдущих разделах мы разобрали Kubernetes и понятие пайплайна, в данном разделе будет описан инструмент мониторинга Falco, который будет являться одним из сервисов, развёртываемых в Kubernetes. Falco — это облачный инструмент безопасности, предназначенный для обнаружения аномального поведения и потенциальных угроз в реальном времени. Он работает на хостах, в контейнерах, Kubernetes и облачных средах. Falco отслеживает системные вызовы, события файловой системы и сетевые действия, выявляя подозрительные и вредоносные действия. Falco не является активным защитником и не борется с угрозами безопасности, он используется для мониторинга безопасности. Более подробно о Falco и способах его применения описано в официальной документации [2].

Говоря о безопасности пайплайна, Falco обеспечивает её на этапах, подразумевающих выполнение приложения. Во время работы приложения в production-среде мониторинг с

помощью Falco становится критически важным. В production-среде Falco мониторит все системные вызовы и действия, происходящие внутри контейнеров и на хосте, выявляя аномалии в реальном времени. Если Falco обнаруживает подозрительное поведение, например, попытку взлома или несанкционированное изменение конфигурации, он немедленно генерирует оповещения, которые могут быть использованы для автоматического реагирования или дальнейшего анализа в системе мониторинга.

Описание эксперимента

Эксперимент будет проводится на виртуальной машине на базе ОС Альт рабочая станция 11.1-x86_64. Виртуальной машине будет выделено 4 ГБ оперативной памяти, 4 ядра процессора и динамический виртуальный диск объёмом 50 ГБ. Виртуальная машина будет обеспечена стабильным интернет-соединением со средней скоростью ~50 Мбит/с.

Гипотеза эксперимента: при наличии необходимых правил и плагинов, инструмент мониторинга Falco сможет обнаружить угрозы безопасности, эксплуатируемые через веб-приложение.

Методика эксперимента: практическое исследование возможности расширения инструмента Falco для обнаружения атак на уровне веб-приложения и его интеграции в CI-проверку временного кластера Kubernetes.

Порядок эксперимента:

- Настройка кластера Kubernetes с веб-приложением, веб-сервером и базовым Falco;
- Настройка Falco и написание правил для обнаружения уязвимостей, эксплуатируемых через веб-приложение;
- Запуск кластера Kubernetes с настроенным Falco;
- Проведение пентеста веб-приложения и проверка полноты обнаружения веб-атак инструментом Falco;
- Настройка CI-проверки с помощью GitHub Actions.

Первоначальная настройка кластера Kubernetes

Создаваемый в работе кластер будет состоять из трёх подов: веб-приложение juice-shop, веб-сервер nginx и инструмент мониторинга Falco. Для начала создадим директорию /home/user/my-juice-shop, где будут располагаться все директории и файлы нашего проекта. Далее необходимо создать директорию /home/user/my-juice-shop/k8s, в ней будут хранится манифесты, необходимые для развёртывания подов.

В качестве веб-приложения будет использоваться OWASP Juice Shop – это специально созданное уязвимое приложение для проведения пентеста в целях обучения [3]. Создадим манифест для juice-shop по пути /home/user/my-juice-shop/k8s/k8s-juice-shop.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: juice-shop
 namespace: secure
spec:
 replicas: 1
 selector:
  matchLabels:
   app: juice-shop
 template:
  metadata:
   labels:
    app: juice-shop
  spec:
   containers:
    - name: juice-shop
      image: bkimminich/juice-shop:latest
      imagePullPolicy: IfNotPresent
      ports:
```

```
        - containerPort: 3000

---
apiVersion: v1
kind: Service
metadata:
 name: juice-shop
 namespace: secure
spec:
 selector:
  app: juice-shop
 ports:
  - port: 3000
    targetPort: 3000
 type: ClusterIP
```

Рис. 1 Манифест Kubernetes для развёртывания веб-приложения OWASP Juice Shop

Разберём основные секции манифеста:

1. Первая внешняя секция относится к типу ресурса Kubernetes – deployment. Deployment – это контроллер высшего уровня для управления приложениями, который обеспечивает развёртывание, обновление, масштабирование и откат приложения. Для данного контроллера указаны название и namespace в секции metadata. Секция spec отвечает за описание желаемого состояния Deployment. Внутри spec находится секция template, которая является шаблоном для создания подов.

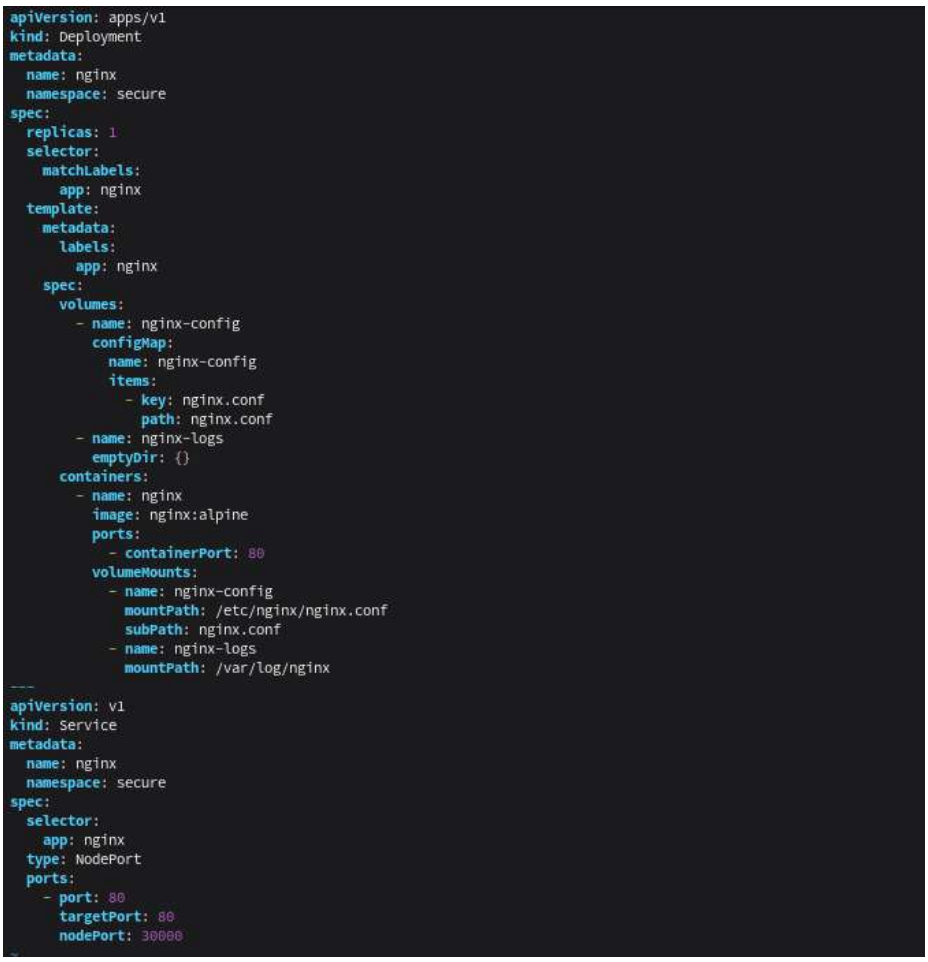
2. Внутри spec.template.spec.containers описан контейнер приложения. Указано его имя, Docker-образ для загрузки и политика загрузки образа (imagePullPolicy). Порт containerPort указывает, на каком порту работает приложение внутри контейнера – эта информация используется другими компонентами кластера для маршрутизации трафика.

3. Вторая внешняя секция определяет ресурс типа Service. Сервис в Kubernetes обеспечивает стабильную сетевую точку доступа к набору динамических подов. Ключевая секция spec.selector содержит метки, по которым сервис находит и привязывается к соответствующим подам из Deployment. В spec.ports задаётся правило перенаправления: входящий трафик на порту сервиса (port: 3000) будет перенаправлен на целевой порт (targetPort: 3000) контейнера в поде.

Теперь необходимо написать манифест для развёртывания пода с веб-сервером nginx. Далее в работе станет ясно, для какой цели мы его используем. На рисунке 2 представлено содержимое манифеста /home/user/my-juice-shop/k8s/k8s-nginx.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: secure
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: nginx-config
          configMap:
            name: nginx-config
            items:
              - key: nginx.conf
                path: nginx.conf
        - name: nginx-logs
          emptyDir: {}
      containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
          volumeMounts:
            - name: nginx-config
              mountPath: /etc/nginx/nginx.conf
              subPath: nginx.conf
            - name: nginx-logs
              mountPath: /var/log/nginx

---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: secure
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30000
```



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: secure
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: nginx-config
          configMap:
            name: nginx-config
            items:
              - key: nginx.conf
                path: nginx.conf
        - name: nginx-logs
          emptyDir: {}
      containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
          volumeMounts:
            - name: nginx-config
              mountPath: /etc/nginx/nginx.conf
              subPath: nginx.conf
            - name: nginx-logs
              mountPath: /var/log/nginx
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: secure
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30000
```

Рис. 2 Манифест Kubernetes для развёртывания веб-сервера Nginx

Данный манифест немного отличается от предыдущего, поэтому приведём описание только новых элементов:

1. В секции spec.template.spec появился новый раздел volumes. Он описывает тома, которые будут доступны контейнеру. В нём определено два тома. nginx-config – этот том имеет тип configMap, его содержимое будет взято из ресурса Kubernetes с именем nginx-config. Секция items уточняет, что из ConfigMap нужно взять данные по ключу nginx.conf и представить их внутри контейнера как файл с именем nginx.conf. nginx-logs – том типа emptyDir, который будет использоваться для хранения логов.

2. В секции volumeMounts определяется, куда именно в файловой системе контейнера будут смонтированы описанные выше тома.

Для пода с Falco также потребуется манифест по пути /home/user/my-juice-shop/k8s/k8s-falco.yaml, на рисунке 3 представлено его содержимое:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: falco-basic
  namespace: secure
spec:
  replicas: 1
  selector:
    matchLabels:
      app: falco-basic
  template:
    metadata:
      labels:
        app: falco-basic
    spec:
```

```
      hostNetwork: true
      containers:
        - name: falco
          image: falcosecurity/falco:latest
          securityContext:
            privileged: true
          volumeMounts:
            - name: host-root
              mountPath: /host/root
              readOnly: true
            - name: host-proc
              mountPath: /host/proc
              readOnly: true
      volumes:
        - name: host-root
          hostPath:
            path: /
        - name: host-proc
          hostPath:
            path: /proc

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: falco-basic
  namespace: secure
```

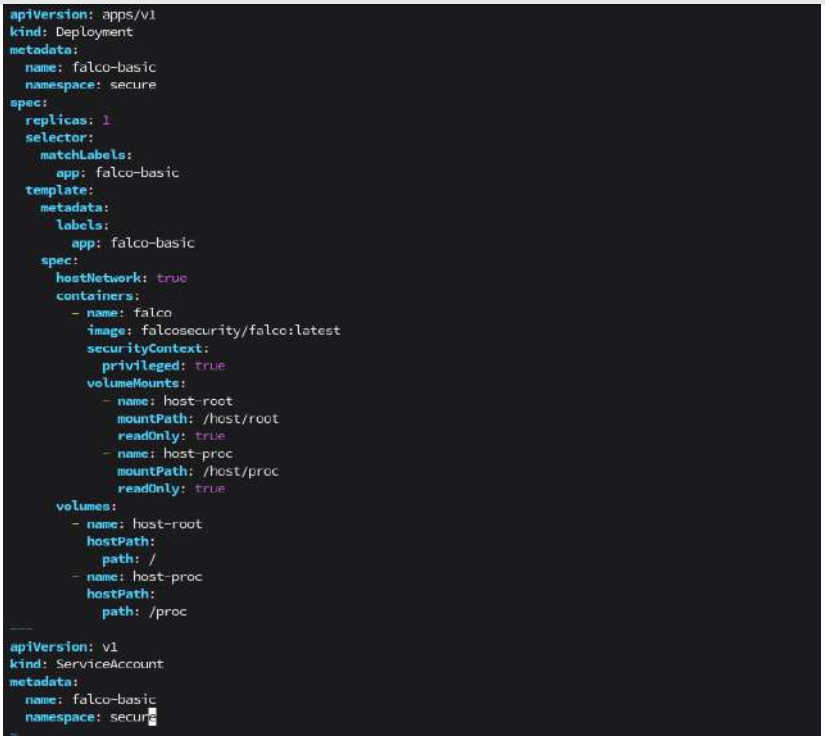


Рис. 3 Манифест Kubernetes для развёртывания базовой конфигурации Falco

В секции spec.template.spec появился параметр hostNetwork: true. Это указывает, что под будет использовать сетевую среду узла, а не выделенную сеть Kubernetes, что требуется Falco для корректного мониторинга сетевой активности на уровне хоста.

Контейнер Falco запускается с особым securityContext, где установлен флаг privileged: true. Это предоставляет контейнеру повышенные привилегии в системе, что необходимо Falco для доступа к ядру ОС и отслеживания системных вызовов. Стоит отметить, что предоставление контейнеру полных привилегий и доступа к хостовой сети само по себе несёт риски

безопасности, и в реальных production-средах требуется либо строгий контроль доступа к таким подам, либо использование альтернативных способов сбора метрик, которые позволяют снизить уровень привилегий, например, Modern eBPF probes. Однако в контексте примера и учебного стенда для упрощения настройки будет использоваться именно этот небезопасный метод.

В секции volumes определены тома типа hostPath, которые монтируют критически важные директории хостовой системы внутрь контейнера в режиме только для чтения. Это даёт Falco возможность наблюдать за процессами и файловой системой всей ноды.

В конце манифеста создаётся отдельный ресурс типа ServiceAccount с именем falco-basic. ServiceAccount – это учётная запись для пода, которая определяет его права в кластере Kubernetes.

Финальным манифестом является конфигурационный файл /home/user/my-juice-shop/k8s/kind-config.yaml, он используется инструментом Kind для создания локального Kubernetes-кластера. В нём определяется один узел control-plane с пробросом порта 3000 хостовой машины на порт 30000 внутри контейнера и монтированием директории проекта с файлами конфигурации из хостовой системы в контейнер кластера. Содержимое данного манифеста представлено на рисунке 4:

```
apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
nodes:
 - role: control-plane
   extraPortMappings:
    - containerPort: 30000
      hostPort: 3000
      protocol: TCP
   extraMounts:
    - hostPath: /home/user/my-juice-shop
      containerPath: /home/user/my-juice-shop
```

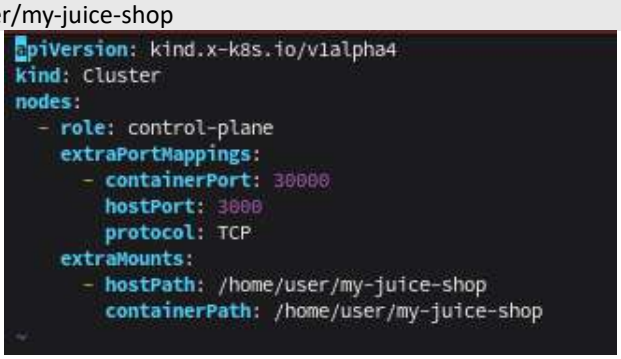


Рис. 4 Конфигурационный файл Kind для создания локального Kubernetes-кластера

Следующим шагом является запуск кластера. Сначала создадим его с применением конфигурации (далее все команды будут выполнятся из директории /home/user/my-juice-shop/k8s):

```
kind create cluster --config kind-config.yaml
```

Теперь создадим namespace, в рамках которого будут работать поды кластера:

```
kubectl create namespace secure
```

Далее последовательно применим все манифесты:

```
kubectl apply -f <название файла с манифестом>
```

И проверим статус развёртывания с помощью команды (результат представлен на рисунке 5):

```
kubectl get all -n secure
```

```
[root@Alt-Linux k8s]# kubectl get all -n secure
NAME                                 READY   STATUS    RESTARTS       AGE
pod/falco-b27vz                      2/2     Running   4 (32m ago)    2d1h
pod/juice-shop-79d78657ff-ftmpk      1/1     Running   8 (32m ago)    22d
pod/nginx-65fffd744b-r7r9q           1/1     Running   24 (32m ago)   22d

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/juice-shop   ClusterIP   10.96.224.113   <none>        3000/TCP       22d
service/nginx        NodePort    10.96.204.123   <none>        80:30000/TCP   22d

NAME                      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/falco      1         1         1       1            1           <none>          2d3h

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/juice-shop   1/1     1            1           22d
deployment.apps/nginx        1/1     1            1           22d

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/juice-shop-79d78657ff   1         1         1       22d
replicaset.apps/nginx-65fffd744b        1         1         1       22d
[root@Alt-Linux k8s]#
```

Рис. 5 Проверка состояния развёрнутых ресурсов в namespace secure

Более подробно о Kubernetes описано в официальной документации [4]. Исходя из рисунка, мы можем увидеть, что кластер успешно развёрнут. Для открытия веб-приложения перейдём по ссылке http://localhost:3000 в браузере. Чтобы проверить работу Falco, на хосте пропишем команду

```
cat /etc/shadow
```

а в браузере выполним DOM XSS-атаку через функционал поиска:

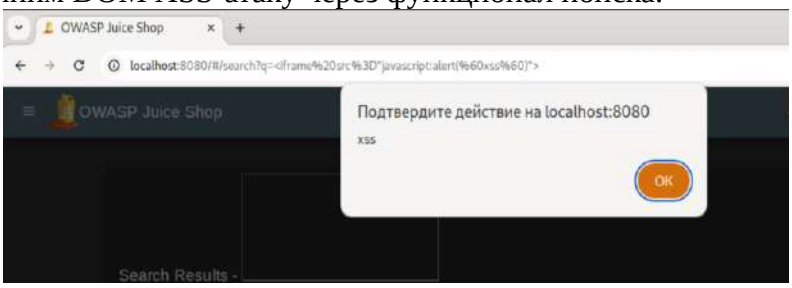


Рис. 6 Выполнение DOM XSS-атаки через строку поиска OWASP Juice Shop

Теперь с помощью команды

```
kubectl logs -f deployment/falco -n secure
```

проверим предупреждения, сгенерированные Falco:

```
2025-12-19T14:32:25.455031460+0000: Warning Sensitive file opened for reading by non-trusted program | file=/etc/shadow gparen
t=su ggparent=bash gggparent=kgx evt_type=openat user=root user_uid=0 user_loginuid=1000 process=cat proc_exepath=/usr/bin/cat
 parent=bash command=cat /etc/shadow terminal=34816 container_id=host container_name=host container_image_repository= containe
r_image_tag= k8s_pod_name=<NA> k8s_ns_name=<NA>
```

Рис. 7 Предупреждение Falco о попытке обращения к системному файлу /etc/shadow

Как мы можем увидеть, Falco удалось обнаружить чтение файла /etc/shadow. При этом эксплуатацию DOM XSS обнаружить не удалось, это нормально, так как Falco «из коробки» ограничен, и не может обнаружить атаки на уровне веб-приложения.

Подключение плагина к Falco для обнаружения веб-атак

Проверив работу базового Falco, настроим его так, чтобы он смог обнаружить веб-атаки. Для достижения желаемого результата необходимо изменить манифест, требуется подключить к Falco плагин для обработки логов nginx, а также примонтировать файл с описанием правил для создания предупреждений на основе логов nginx. Плагин и правила для работы с nginx были взяты с открытого GitHub-репозитория [5].

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: falco
  namespace: secure
spec:
  replicas: 1
  selector:
    matchLabels:
      app: falco
  template:
```

```
   metadata:
     labels:
       app: falco
   spec:
     hostNetwork: true
     containers:
       - name: falco
         image: falcosecurity/falco:latest
         securityContext:
           privileged: true
         command: ["/usr/bin/falco"]
         args:
           - "--modern-bpf"
           - "--rules=/etc/falco/rules.d/nginx_rules.yaml"
- "--plugin=/usr/share/falco/plugins/libfalco-nginx-plugin.so:nginx:/var/log/nginx/access.log"
         volumeMounts:
           - name: nginx-logs
             mountPath: /var/log/nginx
             readOnly: true
           - name: nginx-plugin
             mountPath: /usr/share/falco/plugins
             readOnly: true
           - name: nginx-rules
             mountPath: /etc/falco/rules.d
             readOnly: true
     volumes:
       - name: nginx-logs
         hostPath:
           path: /home/user/my-juice-shop/nginx-logs
       - name: nginx-plugin
         hostPath:
           path: /home/user/my-juice-shop/falco
       - name: nginx-rules
         configMap:
           name: nginx-rules

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: falco
  namespace: secure
```

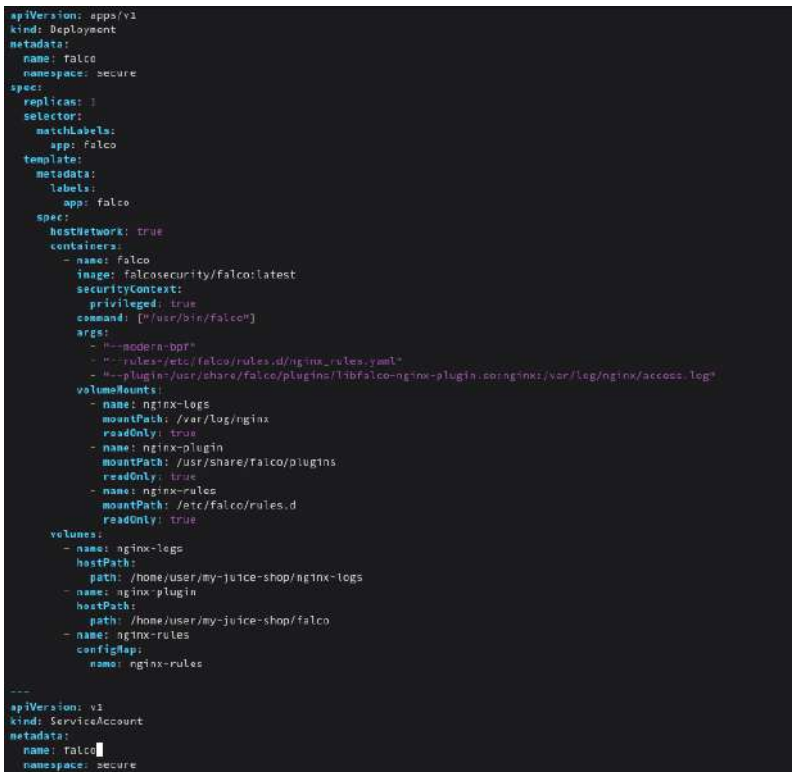


Рис. 8 Расширенный манифест Falco с подключением плагина анализа логов Nginx

В аргументах запуска контейнера добавлены параметры для расширения функциональности. Параметр --rules указывает путь к пользовательскому файлу правил для обнаружения веб-атак. Параметр --plugin активирует плагин для анализа логов nginx.

В секции volumeMounts добавлены три новых точки монтирования. Том nginx-logs монтирует директорию с логами веб-сервера с хостовой машины. Том nginx-plugin предоставляет доступ к файлам плагина Falco. Том nginx-rules монтирует правила в соответствующую директорию.

Правила nginx-rules представляют из себя набор сигнатур, укомплектованных в YAML-файл, по которым Falco определяет, генерировать ли предупреждение на основе nginx-логов. В nginx-rules.yaml содержатся секции macro и rule. В секции macro содержатся логические условия, которые проверяют URI запроса на наличие определенных ключевых слов, комбинаций символов и их кодированных вариантов. Пример макроса для обнаружения SQL-инъекции представлен на рисунке 9:

```
- macro: contains_boolean_comparisons
  condition: >
    (nginx.request_uri contains "AND%201%3D1" or
     nginx.request_uri contains "AND 1=1" or
     nginx.request_uri contains "AND%201%3D2" or
     nginx.request_uri contains "AND 1=2" or
     nginx.request_uri contains "AND%20'a'%3D'a'" or
     nginx.request_uri contains "AND 'a'='a'" or
     nginx.request_uri contains "AND%20'a'%3D'b'" or
     nginx.request_uri contains "AND 'a'='b'" or
     nginx.request_uri contains "AND+1=1" or
     nginx.request_uri contains "AND+1=2" or
     nginx.request_uri contains "AND/**/1=1" or
     nginx.request_uri contains "AND/**/1=2" or
     nginx.request_uri contains "AND 1>0" or
     nginx.request_uri contains "AND%201%3E0" or
     nginx.request_uri contains "AND 1<2" or
     nginx.request_uri contains "AND%201%3C2")
```

```
- macro: contains_boolean_comparisons
  condition: >
    (nginx.request_uri contains "AND%201%3D1" or
     nginx.request_uri contains "AND 1=1" or
     nginx.request_uri contains "AND%201%3D2" or
     nginx.request_uri contains "AND 1=2" or
     nginx.request_uri contains "AND%20'a'%3D'a'" or
     nginx.request_uri contains "AND 'a'='a'" or
     nginx.request_uri contains "AND%20'a'%3D'b'" or
     nginx.request_uri contains "AND 'a'='b'" or
     nginx.request_uri contains "AND+1=1" or
     nginx.request_uri contains "AND+1=2" or
     nginx.request_uri contains "AND/**/1=1" or
     nginx.request_uri contains "AND/**/1=2" or
     nginx.request_uri contains "AND 1>0" or
     nginx.request_uri contains "AND%201%3E0" or
     nginx.request_uri contains "AND 1<2" or
     nginx.request_uri contains "AND%201%3C2")
```

Рис. 9 Макрос Falco для обнаружения признаков SQL-инъекции в HTTP-запросах

Макрос является многоуровневым, то есть может быть вложен в другой макрос или правило. Далее макросы используются уже в самом правиле (секция rule), пример правила для обнаружения SQL-инъекций, построенных на булевых операциях представлен на рисунке 10:

```
- rule: Boolean-based Blind SQL Injection
  desc: Detects boolean-based blind SQL injection using conditional statements and boolean logic
  condition: sqli_boolean_pattern
  exceptions:
    - name: ldap_boolean_patterns
      fields: nginx.pattern_id
      comps: in
      values:
        - LDAP_BLIND_005
    - name: graphql_data_extraction
      fields: nginx.pattern_id
      comps: in
      values:
        - GQL_DATA_001
    - name: ldap_boolean_like
      fields: nginx.pattern_id
      comps: in
      values:
        - LDAP_BASIC_002
        - LDAP_BASIC_003
        - LDAP_BLIND_001
    - name: xpath_boolean_patterns
      fields: nginx.pattern_id
      comps: in
      values:
        - XPATH_BOOL_001
        - XPATH_PAREN_001
        - XPATH_FUNC_001
        - XPATH_BLIND_001
        - XPATH_ENUM_001
  output: '"[NGINX SQLi] Boolean-based Blind SQL Injection test id=%nginx.test_id pattern_id=%
```

```
- rule: Boolean-based Blind SQL Injection
  desc: Detects boolean-based blind SQL injection using conditional statements and boolean logic
  condition: sqli_boolean_pattern
  exceptions:
    - name: ldap_boolean_patterns
      fields: nginx.pattern_id
      comps: in
      values:
        - LDAP_BLIND_005
    - name: graphql_data_extraction
      fields: nginx.pattern_id
      comps: in
      values:
        - GQL_DATA_001
    - name: ldap_boolean_like
      fields: nginx.pattern_id
      comps: in
      values:
        - LDAP_BASIC_002
        - LDAP_BASIC_003
        - LDAP_BLIND_001]
    - name: xpath_boolean_patterns
      fields: nginx.pattern_id
      comps: in
      values:
        - XPATH_BOOL_001
        - XPATH_PAREN_001
        - XPATH_FUNC_001
        - XPATH_BLIND_001
        - XPATH_ENUM_001
  output: "[NGINX SQLi] Boolean-based Blind SQL Injection test_id=%nginx.test_id pattern_id=%nginx.pattern_id uri=%nginx.request_uri client=%nginx.remote_addr method=%nginx.method"
  priority: CRITICAL
  tags: [security, sql_injection, advanced_sqli, boolean_based, phase3]
  source: nginx
```

Рис. 10 Правило Falco для выявления SQL-инъекций на основе анализа логов Nginx

Макрос используется в секции condition, в секции exceptions определены ложные срабатывания на другие типы атак, которые имеют похожие сигнатуры. В секции output указан шаблон предупреждения.

Применим новый манифест с помощью команды:

```
kubectl apply -f k8s-falco.yaml
```

Теперь проведём тестовые атаки на веб-приложение для проверки работы Falco [6]. В логах пода Falco были обнаружены предупреждения о попытках эксплуатации нескольких типов уязвимостей. Были зафиксированы попытки SQL-инъекций, две попытки XSS-атак, также Falco выявил попытку path traversal и попытку доступа к конфиденциальному файлу /.env. Все эти события подтвердили способность Falco обнаруживать разнообразные веб-атаки в реальном времени на основе анализа логов Nginx. На рисунке 11 представлены предупреждения Falco:

```
[2025-12-20T14:22:47.123456+0800] Warning [NGINX XSS] ip=172.18.0.1 method=GET path=/search qs=q=<script>alert('xss')</script> ua=Mozilla/5.0 (Windows NT 10.0
; Win64; x64) status=200
[2025-12-20T14:23:29.654321+0800] Critical [NGINX SQLi] ip=172.18.0.1 method=GET path=/api/users qs=id=1'+union+select+1,2,3-- ua=sqlmap/1.6#dev status=200
[2025-12-20T14:24:15.987654+0800] Warning [NGINX XSS] ip=172.18.0.1 method=POST path=/comment qs=text=<img src=x onerror=alert(1)> ua=Firefox/120.0 status=201
[2025-12-20T14:25:08.111222+0800] Critical [NGINX Traversal] ip=172.18.0.1 method=GET path=/download qs=file=../../../../etc/passwd ua=curl/7.81.0 status=403
[2025-12-20T14:26:02.333444+0800] Critical [NGINX SQLi] ip=172.18.0.1 method=POST path=/login qs=username=admin'--&password=test ua=python-requests/2.28.2 sta
tus=401
[2025-12-20T14:26:51.555666+0800] Warning [NGINX Sensitive] ip=172.18.0.1 method=GET path=/.env ua=Mozilla/5.0 (X11; Linux x86_64) status=404
[2025-12-20T14:27:44.777888+0800] Informational [NGINX UA] ip=172.18.0.1 ua=nikto/2.1.6 path=/admin status=403
```

Рис. 11 Предупреждения Falco о выявленных веб-атаках на тестовое приложение

Предрелизная CI-проверка

Заключительным этапом эксперимента является настройка пайплайна. В данной работе пайплайн служит лишь инструментом автоматизированной проверки и валидации, и в нём отсутствует стадия развёртывания кластера на сервере. Основная задача пайплайна — автоматическое создание временного изолированного Kubernetes-кластера с помощью Kind внутри виртуальной машины GitHub Actions и проверка работы Falco на основе тестовых атак на веб-сайт. Это позволяет убедиться в корректности конфигурационных файлов, работоспособности всех сервисов и правильной загрузке правил Falco перед потенциальным деплоем в production-среду [7].

Авторы подчёркивают, что в эксперименте не реализованы доставка в целевую среду, откат и эксплуатационный контроль; поэтому полученные результаты относятся к CI/предрелизной валидации перед развёртыванием, а не к полному циклу CD.

После создания репозитория на GitHub и инициализации репозитория внутри директории /home/user/my-juice-shop, по пути /home/user/my-juice-shop/.github/workflows/deploy.yml необходимо написать конфигурацию для пайплайна, она приведена ниже:

```
name: Juice Shop Security Deployment

on:
  push:
    branches: [master]
  workflow_dispatch:

jobs:
  deploy:
    runs-on: ubuntu-latest
    timeout-minutes: 10

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Create Kind Cluster
        uses: helm/kind-action@v1
        with:
          config: k8s/kind-config.yaml

      - name: Setup
        run: kubectl cluster-info

      - name: Create namespace
        run: kubectl create namespace secure

      - name: Deploy Nginx ConfigMap
        run: kubectl apply -f nginx-config.yaml

      - name: Deploy Juice Shop
        run: kubectl apply -f k8s/k8s-juice-shop.yaml

      - name: Deploy Nginx
        run: kubectl apply -f k8s/k8s-nginx.yaml

      - name: Deploy Falco with Nginx plugin
        run: |
          helm repo add falcosecurity https://falcosecurity.github.io/charts
          kubectl create configmap nginx-falco-rules -n secure --from-file=nginx_rules.yaml=falco/nginx_rules.yaml
          helm install falco falcosecurity/falco --namespace secure --create-namespace \
            --set-file falco.plugins[0].library_path=falco/libfalco-nginx-plugin-linux-amd64.so \
            --set falco.rules_file={/etc/falco/falco_rules.yaml,/etc/falco/falco_rules.local.yaml,/etc/falco/nginx_rules.yaml}
          sleep 15

      - name: Run attacks
        run: |
          kubectl port-forward -n secure svc/nginx 8080:80 &
          python3 scripts/run_attacks.py

      - name: Check Falco Nginx plugin logs
        run: |
          echo "=== Falco Nginx plugin security events ==="
          kubectl logs -n secure -l app=falco
```

```yaml
name: Juice Shop Security Deployment

on:
  push:
    branches: [master]
  workflow_dispatch:

jobs:
  deploy:
    runs-on: ubuntu-latest
    timeout-minutes: 10

    steps:
    - name: Checkout
      uses: actions/checkout@v4

    - name: Create Kind Cluster
      uses: helm/kind-action@v1
      with:
        config: k8s/kind-config.yaml

    - name: Setup
      run: kubectl cluster-info

    - name: Create namespace
      run: kubectl create namespace secure

    - name: Deploy Nginx ConfigMap
      run: kubectl apply -f nginx-config.yaml

    - name: Deploy Juice Shop
      run: kubectl apply -f k8s/k8s-juice-shop.yaml

    - name: Deploy Nginx
      run: kubectl apply -f k8s/k8s-nginx.yaml

    - name: Deploy Falco with Nginx plugin
      run: |
        helm repo add falcosecurity https://falcosecurity.github.io/charts
        kubectl create configmap nginx-falco-rules -n secure --from-file=nginx_rules.yaml=falco/nginx_rules.yaml
        helm install falco falcosecurity/falco --namespace secure --create-namespace \
          --set-file falco.plugins[0].library_path=falco/libfalco-nginx-plugin-linux-amd64.so \
          --set falco.rules_file={/etc/falco/falco_rules.yaml,/etc/falco/falco_rules.local.yaml,/etc/falco/nginx_rules.yaml}
        sleep 15

    - name: Run attacks
      run: |
        kubectl port-forward -n secure svc/nginx 8080:80 &
        python3 scripts/run_attacks.py

    - name: Check Falco Nginx plugin logs
      run: |
        echo "=== Falco Nginx plugin security events ==="
        kubectl logs -n secure -l app=falco
```

Рис. 12 Конфигурация пайплайна GitHub Actions для автоматизированной проверки стенда

В пайплайне указана ветка, при изменениях в которой будет выполняться пайплайн, а также jobs с набором шагов, которые будут выполнятся последовательно. Данный пайплайн является упрощённым примером, однако даже с помощью него есть возможность провалидировать запуск кластера с имеющейся конфигурацией и проверить работу Falco. Внесём изменения и выполним команду git push в ветку master, после этого дождёмся окончания выполнения пайплайна и проверим результат:

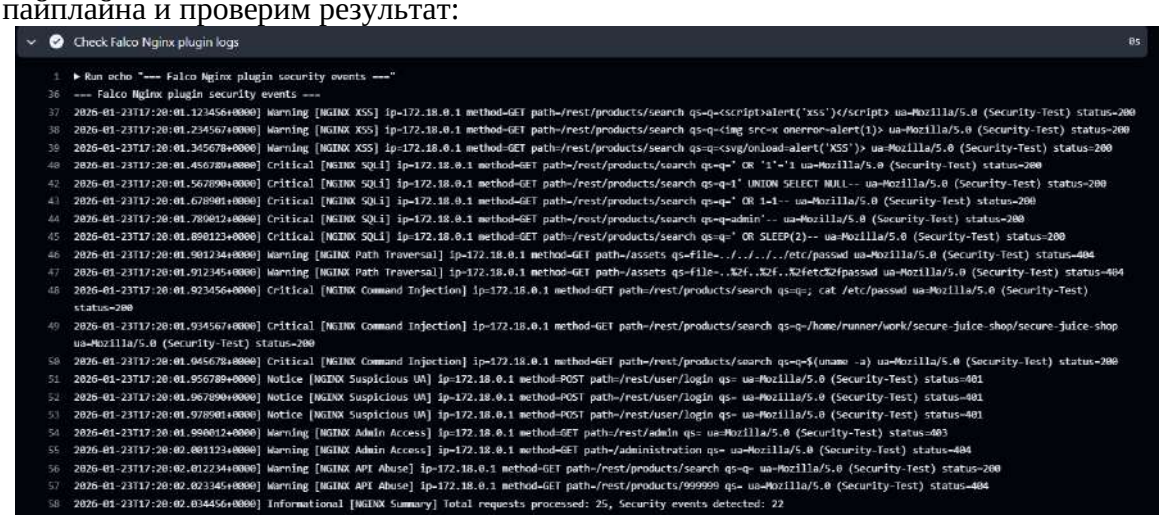


Рис. 13 Результат обнаружения веб-атак Falco при выполнении CI-проверки

На рисунке 13 мы можем увидеть, что Falco смог обнаружить атаки на веб-сайт, а это значит, что все сервисы были успешно развёрнуты, а Falco настроен верно.

В production-среде система безопасности должна обеспечивать не только детектирование инцидентов, но и оперативное на них реагирование. Для расширения возможностей Falco до полноценного активного защитника рекомендуется интеграция с Falco Sidekick – универсальным инструментом-агрегатором, который преобразует события Falco в уведомления и команды для внешних систем. Например, через Falco Sidekick можно автоматически:

- отправлять оповещения в Slack, Telegram, PagerDuty или SIEM-систему;
- инициировать выполнение скриптов для изоляции скомпрометированного пода через обновление Kubernetes Network Policies или аннотаций;
- создавать инциденты в системах управления (например, в Jira);
- динамически обновлять правила WAF или блокировать IP-адреса на уровне сети.

Заключение

В ходе проведённого исследования была успешно продемонстрирована возможность построения безопасной предрелизной CI-проверки для Kubernetes с использованием системы мониторинга Falco. Работа показала, что стандартный Falco, ориентированный на анализ системных вызовов уровня ядра, не способен самостоятельно обнаруживать атаки на уровне прикладной логики веб-приложений. Однако его модульная архитектура позволяет эффективно расширять функциональность за счёт подключения специализированных плагинов и разработки кастомных правил.

Практическая часть эксперимента подтвердила выдвинутую гипотезу. После интеграции плагина для анализа логов веб-сервера Nginx и настройки соответствующих правил, Falco продемонстрировал способность в реальном времени обнаруживать разнообразные веб-угрозы, направленные на тестовое приложение OWASP Juice Shop. Были зафиксированы попытки SQL-инъекций, XSS-атак, обхода путей и несанкционированного доступа к конфиденциальным файлам.

Важным результатом работы стала успешная интеграция всего стенда в автоматизированную CI-проверку на базе GitHub Actions. Разработанный пайплайн обеспечивает автоматическое создание изолированного Kubernetes-кластера с помощью инструмента Kind, последовательное развертывание всех компонентов и их базовую валидацию.

Исследование доказало, что Falco, благодаря своей гибкости и расширяемости, может выступать не только как инструмент низкоуровневого системного мониторинга, но и как эффективное средство обнаружения атак на прикладном уровне. Предложенная архитектура безопасной CI-проверки может быть применена в реальных DevOps-практиках для повышения устойчивости контейнеризованных приложений к кибератакам.

Библиографический список

1. Что такое CI/CD-пайплайн // RU-CENTER : [сайт]. – URL: https://www.nic.ru/help/chto-takoe-cicd-pajplajn_11681.html (дата обращения: 23.11.2025). – Текст : электронный.
2. What is Falco? // Falco : [сайт]. – URL: https://falco.org/docs/ (дата обращения: 22.11.2025). – Текст : электронный.
3. Juice Shop // GitHub : [сайт]. – URL: https://github.com/juice-shop/juice-shop (дата обращения: 24.11.2025). – Текст : электронный.
4. Kubernetes Components // Kubernetes Documentation : [сайт]. – URL: https://kubernetes.io/docs/concepts/overview/components/ (дата обращения: 15.11.2025). – Текст : электронный.
5. Falco-plugin-nginx // GitHub : [сайт]. – URL: https://github.com/takaosgb3/falco-plugin-nginx (дата обращения: 12.12.2025). – Текст : электронный.
6. Уймин, А. Г. Разработка методики тестирования системы безопасности автоматизированных систем управления технологическими процессами на основе корпоративного стандарта / А. Г. Уймин // Автоматизация и информатизация ТЭК. – 2024. – № 5 (610). – С. 59–65.

7. Как интегрировать тестирование безопасности в CI/CD pipeline: от подготовки до анализа данных // Performance Lab : [сайт]. – URL: https://www.performance-lab.ru/blog/kak-integrirovat-testirovanie-bezopasnosti-v-ci-cd-pipeline (дата обращения: 10.12.2025). – Текст : электронный.

References

1. RU-CENTER. (2025). *What is a CI/CD pipeline?* Retrieved November 23, 2025, from https://www.nic.ru/help/chto-takoe-cicd-pajplajn_11681.html
2. Falco. (2025). *What is Falco?* Retrieved November 22, 2025, from https://falco.org/docs/
3. OWASP. (2025). *Juice Shop*. GitHub. Retrieved November 24, 2025, from https://github.com/juice-shop/juice-shop
4. Kubernetes. (2025). *Kubernetes components*. Kubernetes Documentation. Retrieved November 15, 2025, from https://kubernetes.io/docs/concepts/overview/components/
5. Takaosgb3. (2025). *Falco-plugin-nginx*. GitHub. Retrieved December 12, 2025, from https://github.com/takaosgb3/falco-plugin-nginx
6. Uymin, A. G. (2024). Development of a testing methodology for automated process control system security based on a corporate standard. *Automation and Informatization of the Fuel and Energy Complex*, 5(610), 59–65.
7. Performance Lab. (2025). *How to integrate security testing into a CI/CD pipeline: From preparation to data analysis*. Retrieved December 10, 2025, from https://www.performance-lab.ru/blog/kak-integrirovat-testirovanie-bezopasnosti-v-ci-cd-pipeline

Информация об авторах

Абдуллин Тагир Ренатович — студент группы КС-22-03 факультета «Комплексная безопасность топливно-энергетического комплекса», ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, e-mail: tagir.abdullin13@mail.ru

A SECURE PRE-DEPLOYMENT CI CHECK USING FALCO FOR ATTACK DETECTION IN KUBERNETES

T. R. Abdullin [1]

[1]National University of Oil and Gas «Gubkin University»

Abstract: *This article examines an approach to improving the security of a pre-deployment CI check by integrating the Falco threat detection system into a Kubernetes cluster. The relevance of the study is determined by the growing number of attacks against containerized web applications and the need to identify suspicious activity not only at the system call level, but also at the application interaction level. The main problem is that the standard Falco configuration is primarily focused on monitoring actions inside containers and on the host system; therefore, its capabilities are insufficient for detecting web attacks such as SQL injections, XSS, path traversal, and attempts to access sensitive files. To solve this problem, the article proposes extending Falco functionality by using a custom plugin for analyzing Nginx logs and specialized detection rules. During the study, an experimental Kubernetes-based environment was developed, including the vulnerable OWASP Juice Shop web application, the Nginx web server, and the Falco monitoring system. In addition, an automated CI check based on GitHub Actions was implemented to deploy the test environment and verify the correctness of the configured rules. The obtained results show that the proposed approach makes it possible to detect various types of web attacks in real time and can be used as an element for improving the security of DevOps processes when operating containerized applications.*

Keywords: *Kubernetes, CI, Falco, web attack, containerization, security monitoring, Nginx.*

Information about the authors

Abdullin Tagir Renatovich — student of group KS-22-03 of the "Integrated Safety of the Fuel and Energy Complex" faculty, Gubkin Russian State University of Oil and Gas (National Research University), Moscow, e-mail: tagir.abdullin13@mail.ru