**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Институт магистратуры

Кафедра Защищенных систем связи

Дисциплина Технологии обеспечения информационной безопасности больших данных

**ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ**

Обработка событий информационной безопасности

*(тема отчета)*

Направление/специальность подготовки

     10.04.01 - Информационная безопасность

*(код и наименование направления/специальности)*

Студенты:

Фёдорова О.В. ИКТБ-18м

*(Ф.И.О., № группы) (подпись)*

Салита А. С. ИКТБ-18м

*(Ф.И.О., № группы) (подпись)*

Проверил:

         Виткова Л. А.

*(Ф.И.О.) (подпись)*

1. Установите ОС Unix.

Для выполнения лабораторной работы была использована виртуальная Linux Ubuntu 20.04 с характеристиками, указанными на Рисунке 1.

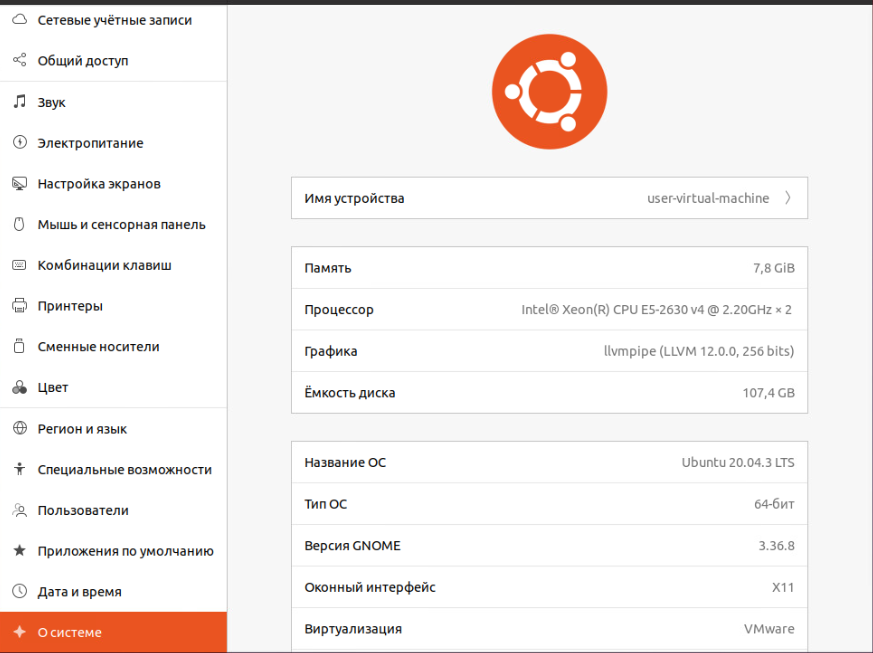


Рисунок Характеристика виртуальной машины

1. Соберите docker compose из нескольких сервисов:

Файл docker-compose.yml выложен на github.

Сервисы, которые были использованы в данной работе:

* node\_exporter (собирает данные о состоянии сервера):
* Prometheus (используемое для мониторинга событий и оповещения, он записывает показатели в реальном времени в базу данных);
* Grafana (выполняет интерактивную визуализацию);
* Falco (мера безопасности, обнаруживающая неожиданное поведение, вторжения и кражу данных в режиме реального времени);
* mysql8 (система управления базами данных);

После запуска docker-compose, стартуют все установленные контейнеры, что можно увидеть на Рисунке 2:

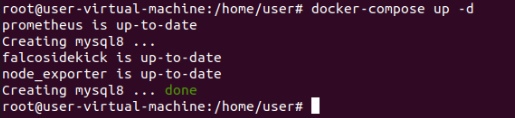


Рисунок Запуск контейнеров

Для более удобной работы с контейнерами используется Portainer.io. Его интерфейс приведен на Рисунке 3. Так же данный рисунок отображает информацию после запуска всех контейнеров. При его помощи можно удобно просматривать запущенные контейнеры, все контейнеры, находящиеся на данной машине, а также выполнять различные действия с ними.

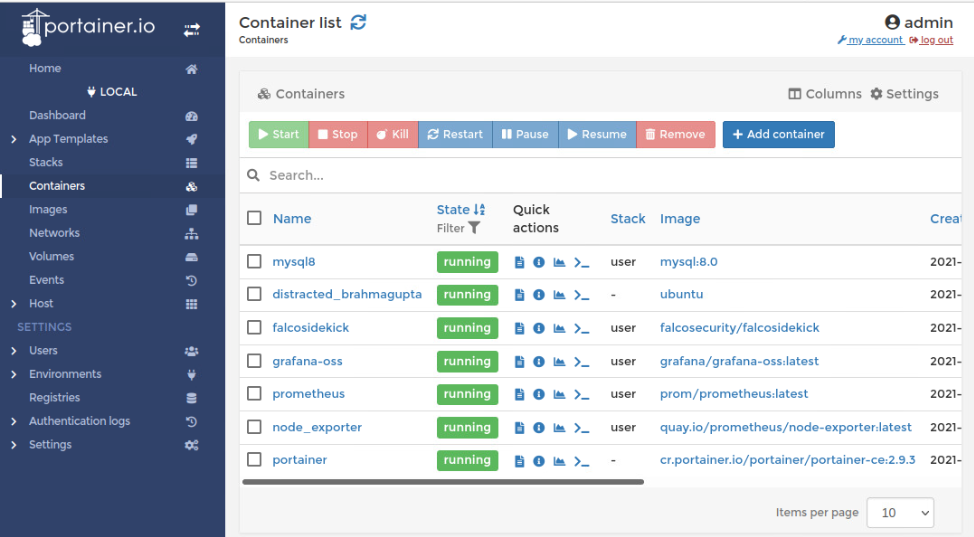


Рисунок Portainer.io

1. Установите Falco security в docker в режиме без eBPF, добавьте правила для обнаружения событий информационной безопасности.

Устанавливаем и запускаем falco. Результат представлен на Рисунке 4.

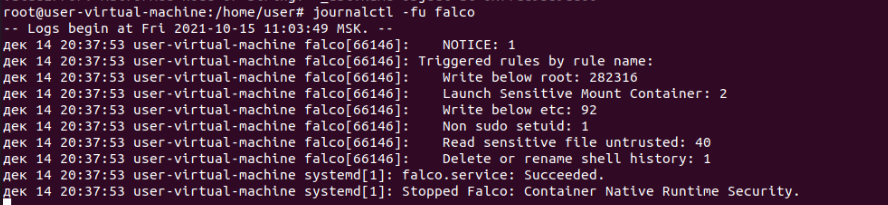
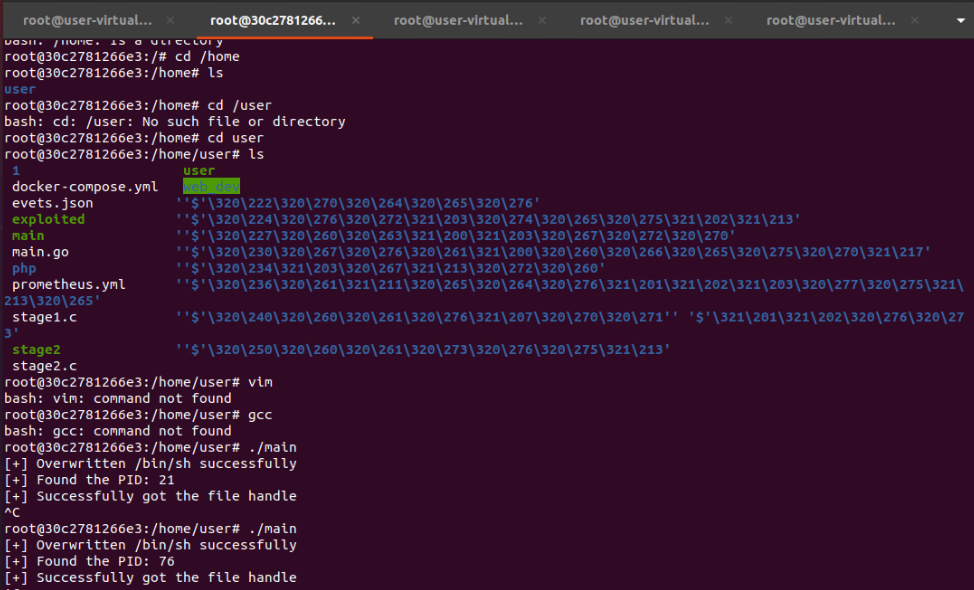


Рисунок Запуск falco

Правила для работы falco прописаны в файле falco\_rules.yaml, который выложен на github.

1. Запустите Falco, контейнеры и поэксплуатируйте уязвимости контейнеров.

Была использована уязвимость: <https://github.com/Frichetten/CVE-2019-5736-PoC>



В результате в логах фалко мы получаем ошибку:

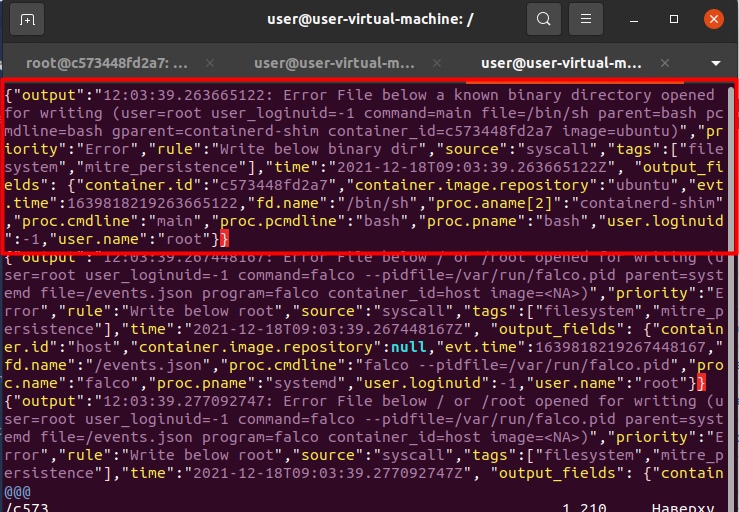


Рисунок Ошибка в логах фалко при использование уязвимости

1. Убедитесь, что Falco обнаруживает попытку побега из контейнера и посмотрите журнал Falco

На Рисунке 6 в 9 строке зарегистрировано событие, не соответствующее норме.

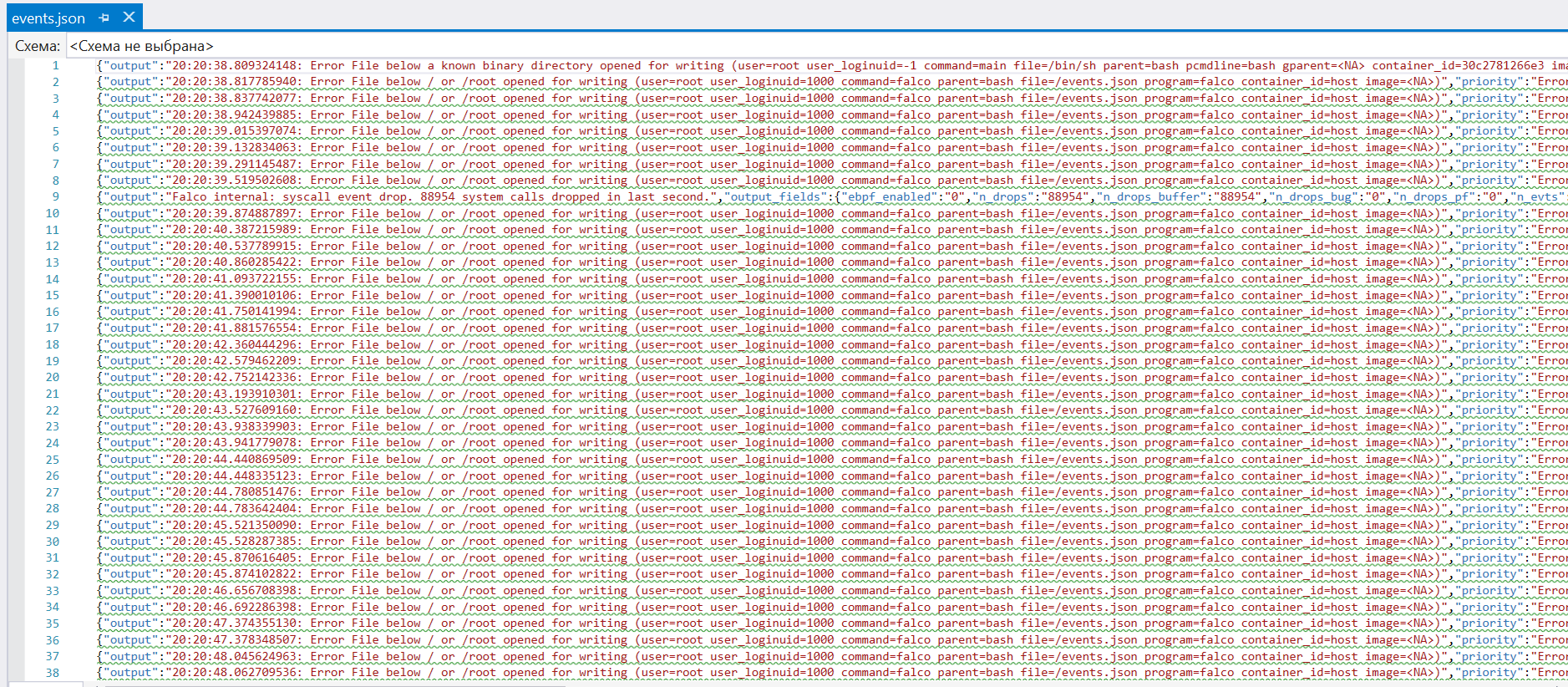


Рисунок Обнаружение ошибок

1. Напишите скрипт для выгрузки журнала с событиями из Falco в JSON (есть на сайте пример в разделе ALERT)

Для экспорта логов в текст, json изменим конфигурацию falco, как представлено на Рисунках 5 и 6:

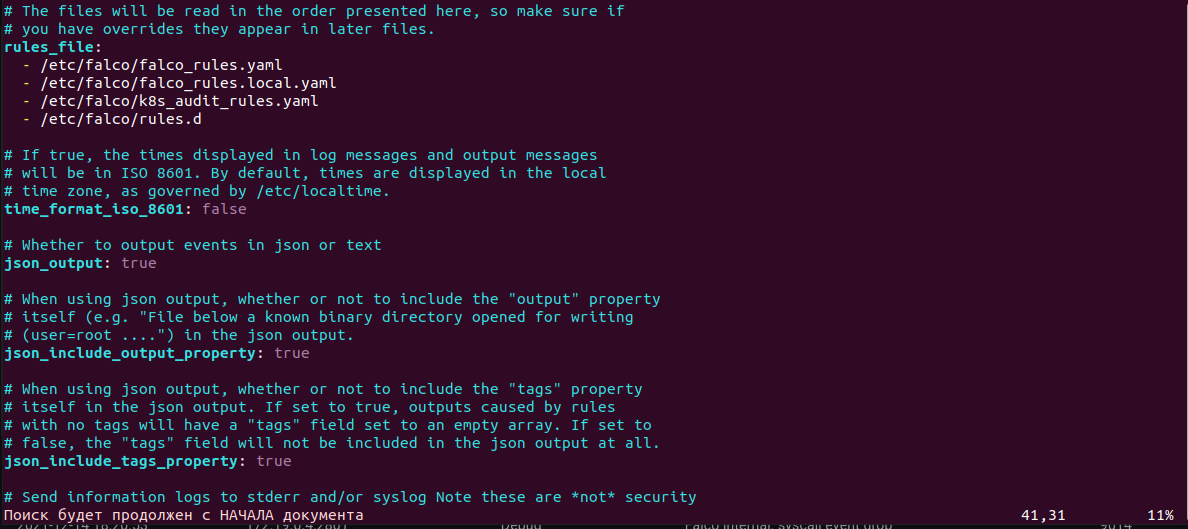


Рисунок Изменение конфигурации falco

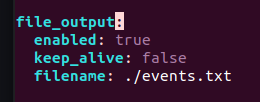


Рисунок Изменение конфигурации falco

1. Установите Falco -экспортер

В качестве falco-экспортера был установлен Falco sidekick в качестве контейнера.

Настройка:

В falco.yaml прописываем:

json\_output: true

json\_include\_output\_property: true

http\_output:

enabled: true

url: "http://localhost:2801/"

Экспортер в Prometheus включен по умолчанию.

1. Установите Prometheus and Grafana

Для установки Prometheus, добавим его в наш докер контейнер:

Часть конфигурации Prometheus:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: "falcosidekick"

static\_configs:

- targets: ["172.19.0.5:2801"]

- job\_name: "prometheus"

static\_configs:

- targets: ["localhost:9090"]

- job\_name: node

static\_configs:

- targets: ['172.19.0.2:9100']

Пример работы Prometheus можно увидеть на Рисунке 7. Доступ к нему осуществляется через порт 8080.

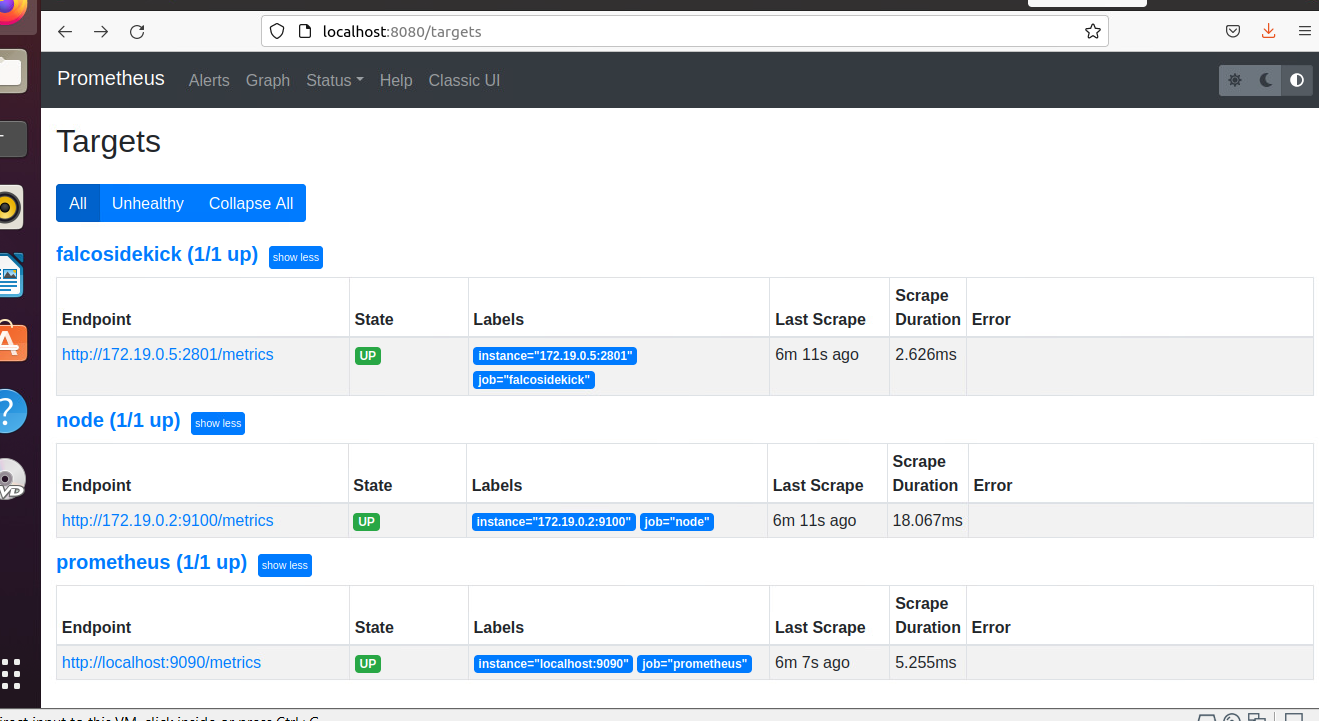


Рисунок Prometheus

Установка Grafana:

Добавьте ключ Grafana GPG, который позволяет устанавливать подписанные пакеты:

sudo apt-get install -y gnupg2 curl software-properties-common

curl https://packages.grafana.com/gpg.key | sudo apt-key add -

sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"

sudo apt install grafana

Запуск Graphana:

Затем, после успешной установки, используем systemctl для запуска сервера Grafana:

sudo systemctl enable --now grafana-server

sudo systemctl start grafana-server

HTTP-порт Grafana по умолчанию — 3000, вам нужно разрешить доступ к этому порту на брандмауэре:

sudo ufw enable

sudo ufw allow 3000/tcp

По умолчанию Grafana будет доступна через HTTP-порт 3000. Открываем браузер и вводим свои учетные данные администратора (имя пользователя и пароль) для Grafana, в этом случае используйте « admin » в обоих случаях, затем нажимаем логин. <http://my-ip-address:3000>

В результате у нас успешно установлена Grafana. Интерфейс программы приведен на Рисунке 9.

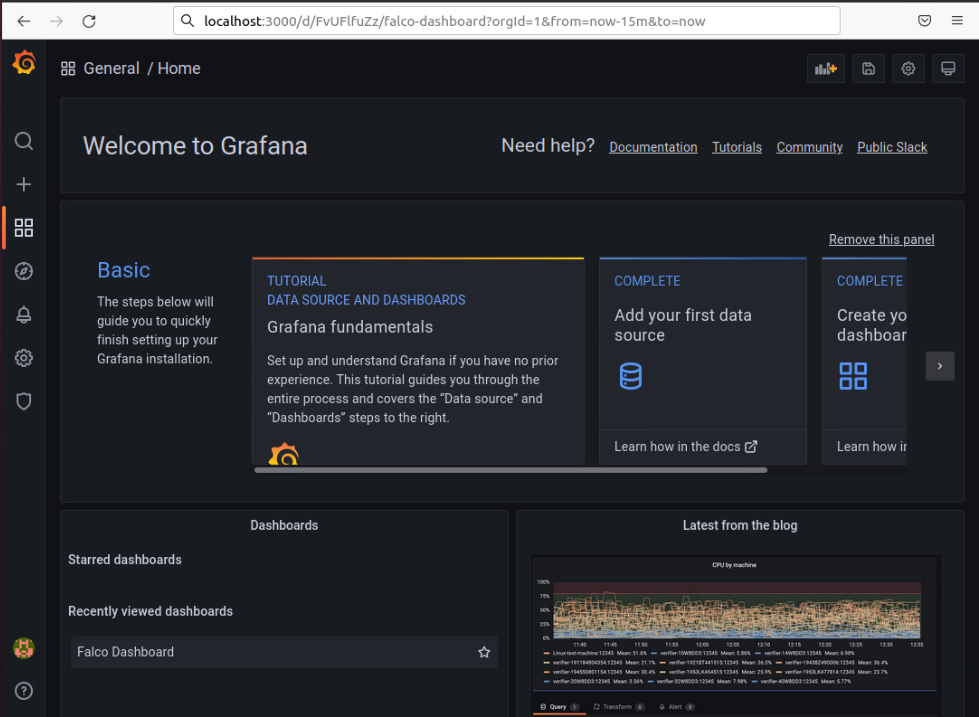


Рисунок Grafana

1. Свяжите Falco при помощи экспортера с Prometheus

Связь была описана в etc/prometheus.yml :

- job\_name: "falcosidekick"

static\_configs:

- targets: ["10.10.0.5:2801"]

- job\_name: "prometheus"

static\_configs:

- targets: ["localhost:9090"]

- job\_name: node

static\_configs:

- targets: ['10.10.0.2:9100']

1. В Grafana настройте dashboard для данных из Prometheus

Для создания dashboard было использовано готовое решение, шаги реализации приведены ниже:

1. Щелкните заголовок графика, затем нажимаем "Изменить".
2. На вкладке "Показатели" выберите источник данных Prometheus (справа внизу).
3. Введите любое выражение Prometheus в поле "Запрос", используя поле "Метрика" для поиска метрик с помощью автозаполнения.
4. Чтобы отформатировать названия условных обозначений временных рядов, используйте ввод "Формат условных обозначений". Например, для отображения только меток method и status возвращенного результата запроса, разделенных тире, можно использовать строку формата условных {{method}} - {{status}} обозначений.

Загружаем данные из Prometheus в Graphana с настройками, указанными на Рисунке 11.

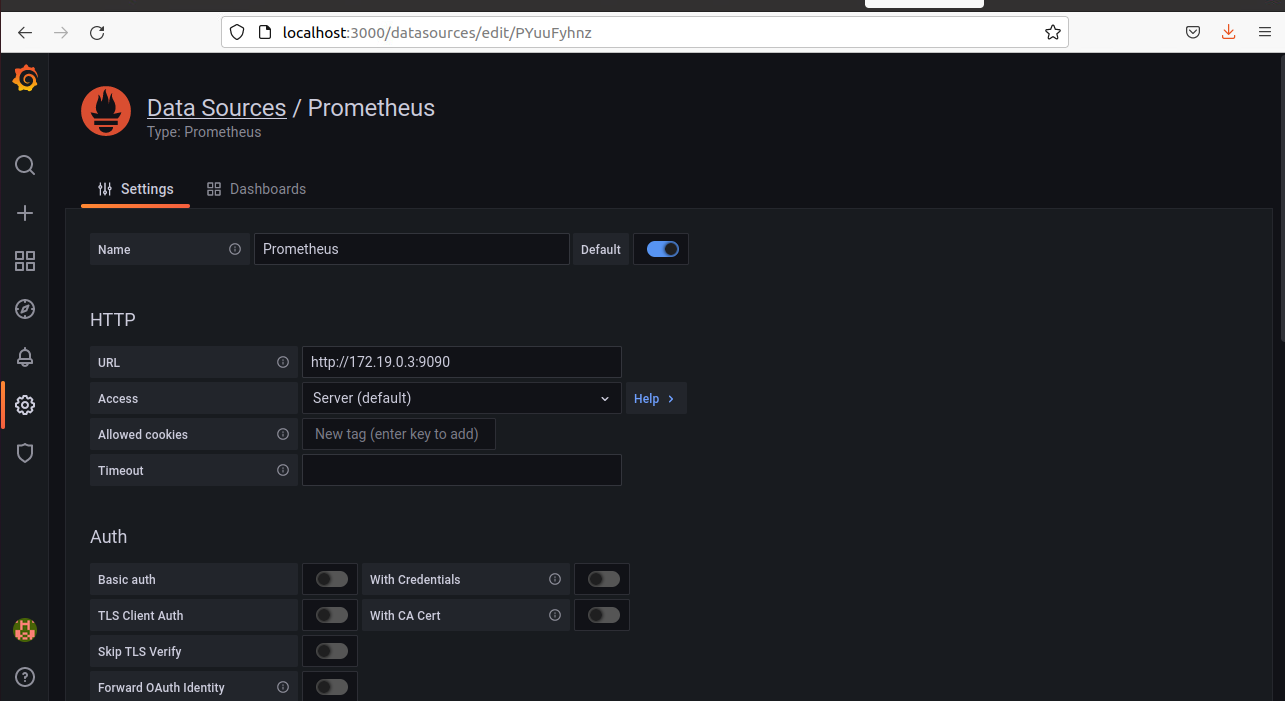


Рисунок Пример конфигурации dashboard

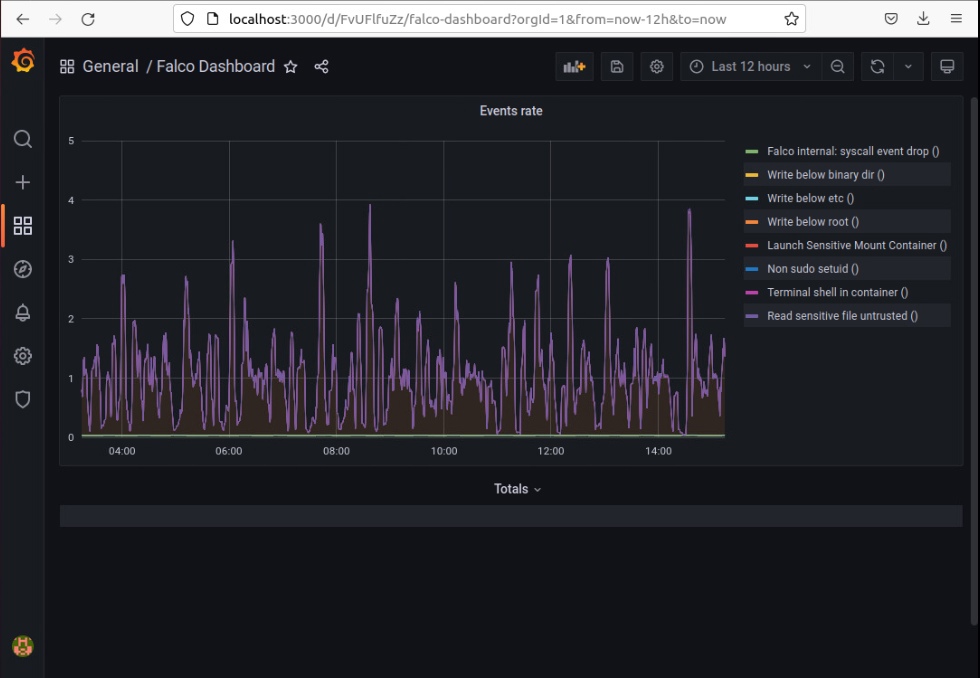


Рисунок Dashboard

1. Обратите внимание на то, как события безопасности отражаются в Prometheus

Пример графа отображающего события безопасности для Prometheus, представлены на Рисунке 13. Мы получаем граф, соответствующий событиям из falco.

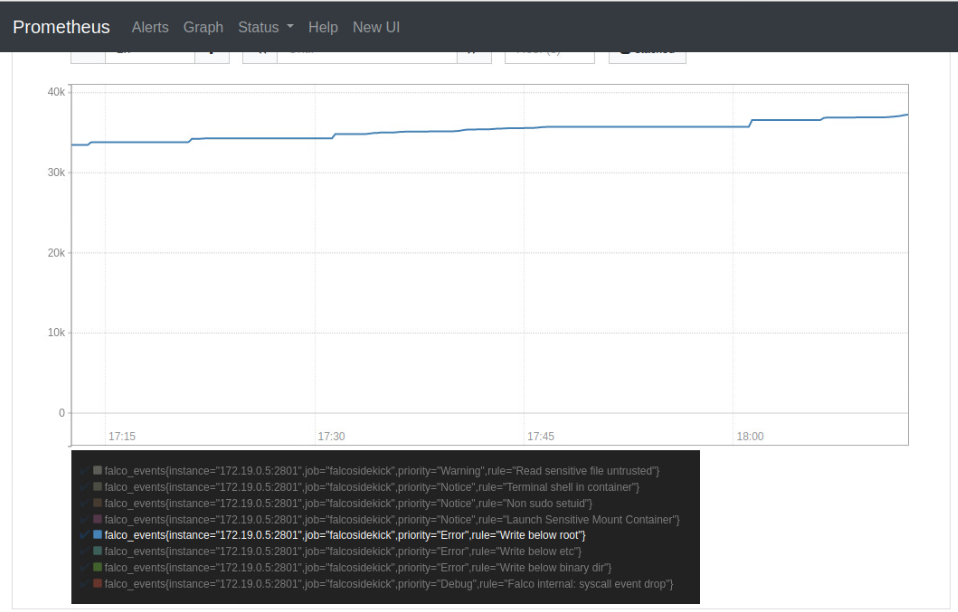


Рисунок Отображение событий в Prometheus

Для сравнения приведена визуализация, получаемая с помощью Graphana:

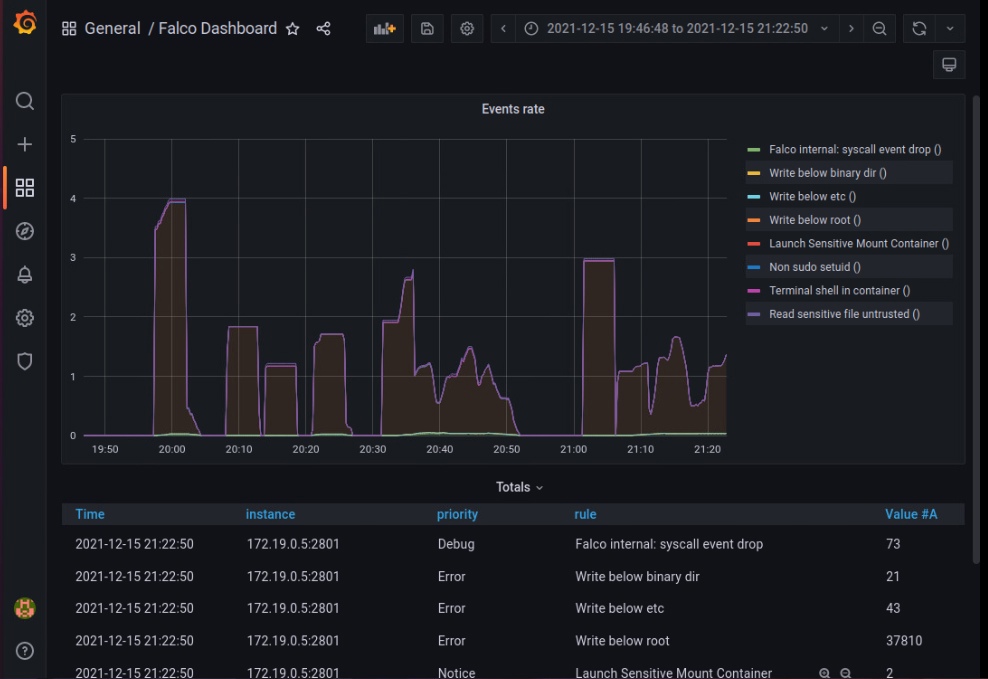


Рисунок Отображение событий в Graphana

1. Почитайте документацию по Prometheus и выберите плагины для анализа производительности узла, для экспорта файлов журналов с узла

Для экспорта файлов с узла используется Nodeexporter.

Он node\_exporterпредназначен для мониторинга хост-системы. Не рекомендуется развертывать его в качестве контейнера Docker, поскольку для этого требуется доступ к хост-системе.

В ситуациях, когда требуется развертывание Docker, необходимо использовать некоторые дополнительные флаги, чтобы разрешить node\_exporterдоступ к пространствам имен хостов.

Имейте в виду, что любые некорневые точки монтирования, которые вы хотите отслеживать, необходимо будет смонтировать в контейнере.

Если вы запускаете контейнер для мониторинга хоста, укажите path.rootfs аргумент. Этот аргумент должен соответствовать пути в привязке-монтировании корневого узла. node\_exporter будет использоваться в path.rootfs качестве префикса для доступа к файловой системе хоста.

docker run -d \

--net="host" \

--pid="host" \

-v "/:/host:ro,rslave" \

quay.io/prometheus/node-exporter:latest \

--path.rootfs=/host

Для Docker compose необходимы аналогичные изменения флага.

services:

node\_exporter:

image: quay.io/prometheus/node-exporter:latest

container\_name: node\_exporter

command:

- '--path.rootfs=/host'

network\_mode: host

pid: host

restart: unless-stopped

volumes:

- '/:/host:ro,rslave'

1. Запустите еще раз эксплойт и сравните то, что видит Falco, и что видит Prometheus при помощи его плагинов.

В результате работы falco мы получаем events.json файл, который выглядит следующим образом Рисунок 15. Он содержит в себе информацию о времени совершения события, пользователя, его наименование и так далее.

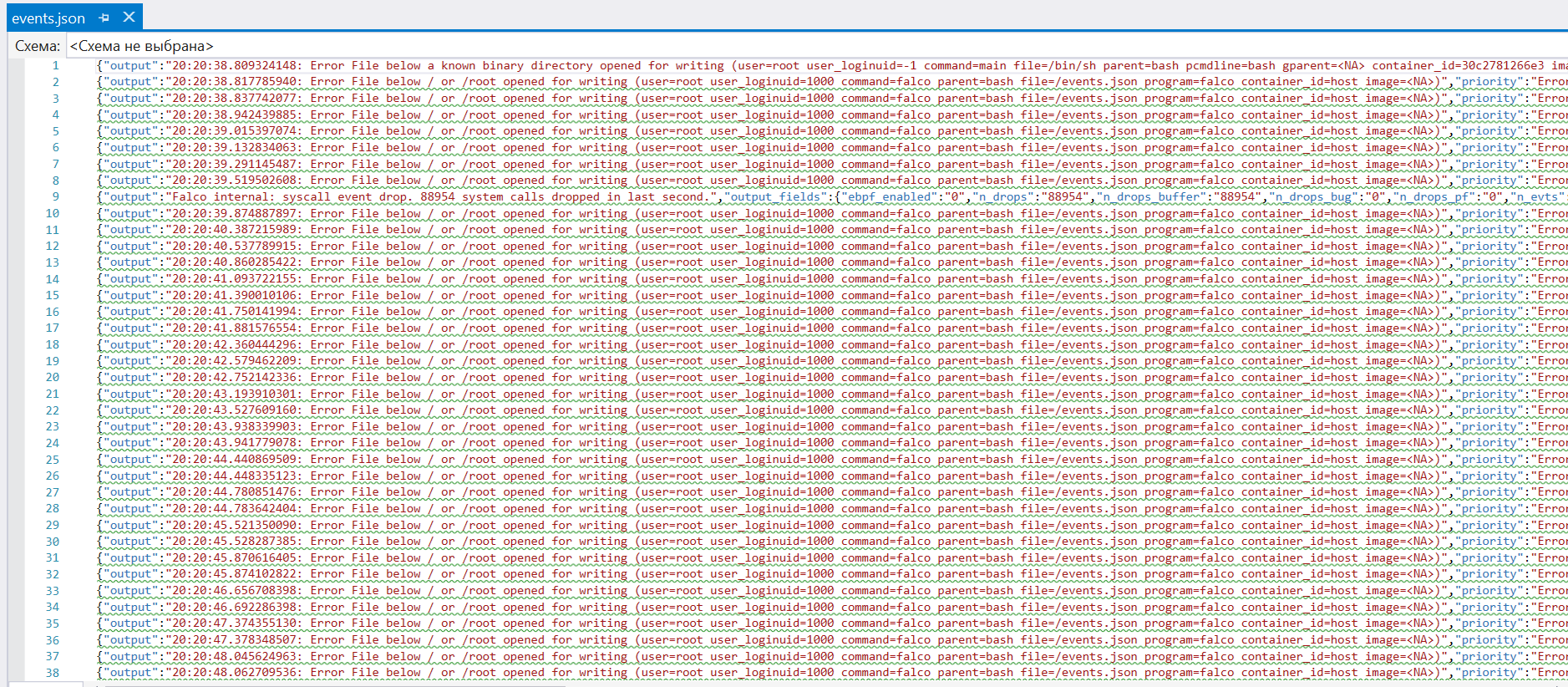


Рисунок Файл получаемый от falco

Данные в прометеус представлены в виде графа:

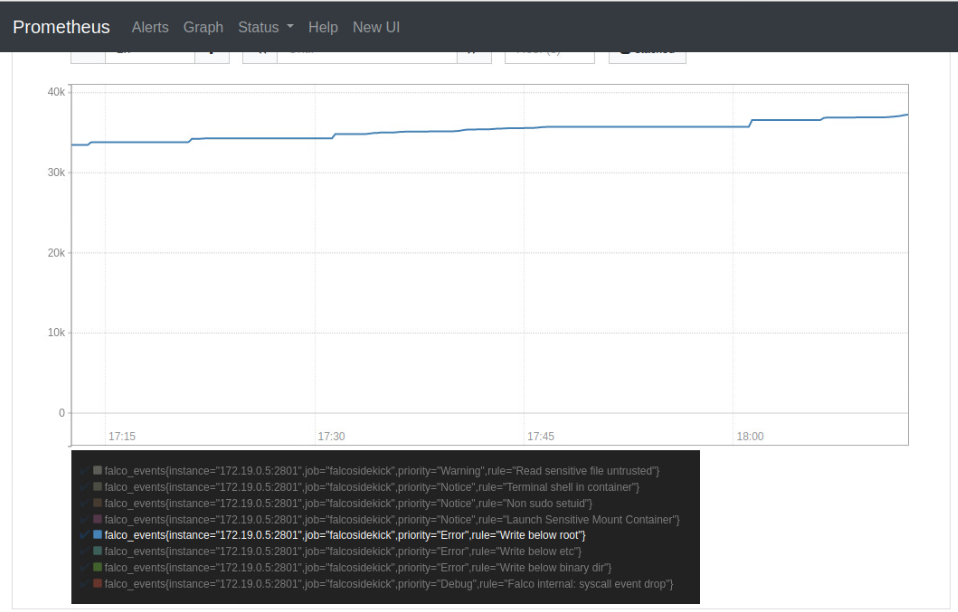


Рисунок Данные прометеус

В прометеусе и графане описываются события, представленные на Рисунке 17.

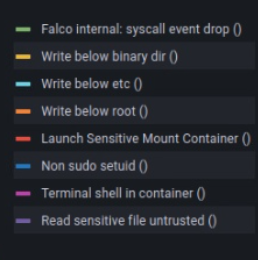


Рисунок События

Они представляют собой:

* Отброшенные системные вызовы;
* Запись в бинарный каталог;
* Запись в каталог /etc;
* Запись от имени root;
* Запуск контейнера;
* Команды не суперпользователя;
* Оболочка терминала в контейнере;
* Чтение конфиденциальных файлов без доверия;

1. Обучите модели машинного обучения (методы классификации)
2. Протестируйте модели машинного обучения
3. Подготовьте курсовой проект, в котором будут описаны все шаги из данного задания, все этапы выполнения.
4. На GitHub выложите:
   1. docker compose,
   2. docker file: Falco, Prometheus, Grafana
5. генератор набора данных
6. Модели машинного обучения
7. Результаты тестирования машинного обучения с визуализацией при помощи Matplotlib
8. Сравнительное описание для Falco как коллектора и Prometheus (с плагинами)
9. В курсовом необходимо описать требования к системе (ОС, количество ядер, оперативная память)