

Φ-APISEC

РУКОВОДСТВО АДМИНИСТРАТОРА

SΦUAD

www.squadsec.ru

- ✦ Введение
- ✦ Разворачивание системы
 - Дополнительные инфраструктурные компоненты
- ✦ Конфигурирование системы
 - Принцип конфигурирования
 - Использование переменных окружения
 - Использование сервера конфигурации
 - Общие параметры конфигурации (файл application.yml)
 - Параметры, специфичные для конкретного окружения
 - Конфигурирование модулей
 - srv_request_logger
 - Раздел spring
 - Раздел application
 - Настройка алертов
 - Дополнительное условие: url
 - Дополнительное условие: analyzers
 - Настройка Маскирования
 - Настройка Ограничения хранения событий
 - Раздел inventory
 - Раздел db
 - Партиционирование
 - ui
 - srv_config_server
 - http-sniffer
 - envoy_alc
 - Наследование параметров
 - Определение роута при использовании Envoy
 - Определение роута при использовании Nginx
 - Определение роута при использовании http-sniffer
 - lua-скрипты
 - Настройка анализаторов
 - srv-ml-gateway
 - Конфигурация отдельных модулей.
 - class-analyzer
 - srv_signature_analyzer
 - srv_swagger_analyzer
 - Префикс api-gateway.
 - srv_statistic_analyzer
 - Режимы работы
 - Режимы агрегации (группировки)
 - Настройка параметра: filter
 - Исключения анализаторов
 - Конфигурирование ролей и привилегий
 - ✦ Мониторинг работы системы
 - Healthcheck
 - ✦ Создание ключа для git-репозитория
 - ✦ Настройка Envoy
 - ✦ Настройка Nginx
 - ✦ Шифрование/дешифрование паролей
 - Инструкция jasypt-service

- Описание
- Запуск с помощью Docker Compose
- Запуск с помощью Docker
- Использование
- ⊕ Работа в боевом режиме
 - Инфраструктурные компоненты
 - Kafka
 - Postgres
 - Git репозиторий
 - LDAP
 - Containers manager
- ⊕ Интеграционное API
 - Назначение
 - Обязательные параметры запроса:
 - Параметр fromId
 - Параметр limit (не является обязательным)
 - Авторизация
 - Примеры валидных запросов
 - Получение событий за период
 - Коды ответов
 - Особенности реализации

Введение

Документ предназначен для использования в процессе эксплуатации решения Q-APISec и содержит описание настроек модулей, возможностей мониторинга работы и другие необходимые для администрирования сведения.

Разворачивание системы

Модули поставляются в формате docker-образов. Для запуска возможно использование docker, docker-compose и других подобных средств.

Дополнительные инфраструктурные компоненты

Для работы системы потребуются следующие инфраструктурные компоненты.

- ✚ git-репозиторий
- ✚ kafka
- ✚ postgres
- ✚ ldap

Для тестового окружения возможно использование docker-контейнеров (входят в поставку системы).

Для "боевого" применения docker-контейнеры использовать не рекомендуем.

Базу данных Postgres использует сервис srv_request_logger.

Конфигурирование системы

Все конфигурационные файлы системы находятся в git-репозитории. Переопределение параметров возможно при старте модуля через переменные окружения.

Принцип конфигурирования

Использование переменных окружения

Ключевые параметры передаются модулю при старте через переменные окружения.

Переменная	Назначение	Пример
SPRING_PROFILES_ACTIVE	Профиль конфигурации	"testcontainers"
CONFIG_URL	URL сервера конфигурации	"http://srv-config-server:8888"
JAVA_TOOL_OPTIONS	Параметры JVM	"-Xms256m -Xmx256m -XX:+UseG1GC -verbose:gc -XX:MaxGCPauseMillis=10"

Использование сервера конфигурации

Большинство модулей (кроме UI) при старте подключаются к модулю srv_config_server

для получения полной конфигурации. URL сервера конфигурации задается через переменную окружения (CONFIG_URL).

Сервер конфигурации формирует полный список параметров модуля следующим образом.

Для формирования используются аргументы, полученные от приложения: "имя приложения", "профиль конфигурации". Сервер конфигурации вычитывает из git базовый файл application.yml. В нем описана общая для всех модулей конфигурация. К этому файлу добавляется файл для запрашиваемого модуля. Имя файла определяется так: "имя приложения"- "профиль конфигурации".yml. Если какой-то параметр определен в обоих файлах, то приоритет у параметра из файла "имя приложения"- "профиль конфигурации".yml.

Часть параметров модулей применяются только при старте, переопределение некоторых параметров возможно во время работы модуля.

В качестве сервера конфигурации используется Spring Cloud Config
<https://docs.spring.io/spring-cloud-config/docs/current/reference/html/>

Общие параметры конфигурации (файл application.yml)

В этом файле находятся общие для всех модулей параметры, использующих сервер конфигурации.

Для мониторинга работы системы используется стандартный компонент Spring Actuator
<https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html#actuator.endpoints.enabling>

Раздел конфигурационного файла "management" относится к настройкам Actuator. Предполагается, что администраторы системы не будут вносить изменения в этот раздел без согласования с разработчиком системы.

Раздел конфигурационного файла "application". В этом разделе описаны параметры kafka, используемые в системе.

Префикс параметров consumer kafka: application.kafka.consumer
 auto-offset-reset = earliest
 concurrency = 4 - сколько consumer-ов и producer-ов будет создано.

Префикс параметров топиков kafka: application.kafka.topics

Параметр конфигурации	Назначение	Значение по умолчанию
user-request-log	результаты проверки запроса и ответа	"userRequestLog"

Примечание.

В топик userRequestLog одним сообщением отправляются результаты проверки запроса, ответа и accessLog.

Раздел "application.rest" задает префикс внутреннего api. Изменения не предполагаются. Раздел "logging" задает уровень логирования в системе.

Параметры, специфичные для конкретного окружения

Имя окружения задается следующим образом. Имя конфигурационного файла application-test.yml, где test – имя окружения.

Раздел application.kafka.jaas задает аутентификационные данные для подключения к кластеру kafka.

Параметр конфигурации	Назначение
enabled: true	включить механизм аутентификации
login-module: "org.apache.kafka.common.security.plain.PlainLoginModule"	используемый модуль
options.username	логин пользователя
options.password	пароль пользователя
properties.security.protocol: "SASL_PLAINTEXT"	параметры аутентификации
properties.sasl.mechanism: "PLAIN"	параметры аутентификации

Раздел application.kafka.bootstrap-servers - хост:порт bootstrap server-a Kafka.

Раздел application.git - подключение к git - репозиторию.

Параметр	Назначение	Пример
url	Путь к репозиторию	ssh://git@github.com/api-security-gateway/rules-config.git
private-key-path	Путь к приватному ключу	~/.ssh/id_ecdsa

Ключевое слово для расшифровки паролей.

jasupt.encryptor.password - используется для расшифровки паролей, которые хранятся в конфигурации в зашифрованном виде.

Конфигурирование модулей

Конфигурация конкретного модуля состоит из объединения файлов application.yml (описанного выше) и файла "имя приложения"- "профиль конфигурации".yml, в этом файле описываются специфические параметры, характерные для конкретного модуля или специфические значения параметров. Специфические параметры описываются ниже.

Общие для всех модулей параметры

Параметр конфигурации	Назначение	Значение по умолчанию
-----------------------	------------	-----------------------

server.port	Порт встроенного http-сервера. Сервер отдает метрики и обрабатывает внутренние api-вызовы, если такие есть в модуле.	В каждом модуле свое, например, 8181
-------------	--	--------------------------------------

Условные обозначения.

Параметры в виде массивов обозначаются символом: [].

srv_request_logger

Базовый модуль интерфейса администратора. Логирует запросы/ответы, рассчитывает статистику, содержит функции инвентаризации API.

Раздел spring

Параметр конфигурации	Назначение
datasource.url	URL подключения к базе данных хранения запросов (работа приложения) Пример: jdbc:postgresql://localhost:5441/httpRequestLogDb?stringtype=unspecified
datasource.username	пользователь
datasource.password	пароль
flyway.enabled	флаг включения миграций базы данных. По умолчанию true
ldap.urls	URL подключения к серверу LDAP. Пример: ldap://localhost:1389
ldap.base	База для подключения: Пример: ou=users,dc=example,dc=org,dc=users
ldap.username	пользователь
ldap.password	пароль

Раздел application

Параметр конфигурации	Назначение
-----------------------	------------

analyzers	Список активных анализаторов. Пример: - signature-analyzer - swagger-analyzer - statistic-analyzer - class-analyzer
Авторизация	
jwt.secret	Секретное слово для формирования JWT токена. Пример: "secretString"
jwt.life-time.access	Время жизни access-токена (сек). Пример: 86400
jwt.life-time.refresh	Время жизни refresh-токена (сек). Пример: 604800
Алерты	
alerts.waitRequestCheck	Максимальное время ожидания ответа от анализатора (сек). Пример: 30000
alerts.targets	Список обработчиков событий
alerts.targets[].name	Название обработчика (rsyslog)
alerts.targets[].properties.transport	Транспортный протокол. UDP или TCP
alerts.targets[].properties.hostname	адрес rsyslog сервера
alerts.targets[].properties.port	порт rsyslog сервера. По умолчанию 10514

alerts.rules[]	Список правил для алертов
alerts.rules[].route	Роут правила
alerts.rules[].url	Фильтр по URL (опционально) Пример: /
alerts.rules[].status	Фильтр по статусу ответа от анализатора(ов), ОК или FAIL
alerts.rules[].level	Уровень логирования в обработчике (rsyslog)
alerts.rules[].target	Обработчик. Пример: rsyslog
Маскирование	
mask.enabled	Параметр указывает, будет ли использоваться маскирование
mask.class-analyzer	Параметр указывает, использовать ли данные классификатора.
mask.data-class	Категории данных для маскирования тела запроса и тела ответа. Пример: CREDIT_CARD, FIO
mask.rules	Правила маскирования
mask.rules.in	Части запроса/ответа на которое будут применяется маскирования. Пример: response_payload,url,request_header,response_header,request_payload
mask.rules.regex	Правило маскирования на основе регулярных выражений
mask.rules.header	Заголовок для маскирования
Отчеты	
report.git-repository	URL гит-репозитория для сохранения отчетов (опционально)

report.file-repository	Путь до директории для сохранения отчетов (опционально)
report.private-key-path	Путь до директории с приватными ssh ключами (нужно для авторизации для report.git-repository)
Ограничение хранения событий	
filter.cache.maxSize	Максимальный размер кэша. опционально (по умолчанию 1000000)
filter.defaultRoute.enabled	Включен ли фильтр для роута (по умолчанию true)
filter.defaultRoute.limit.duration	Длительность временного окна или размер size (одна из опция обязательна) по стандарту ISO 8601 duration format . Пример: PT10S
filter.defaultRoute.limit.capacity	Максимальное кол-во запросов
filter.defaultRoute.criteria	Критерий уникальности запроса (по умолчанию routeld). Пример: method,url
filter.routes	Список роутов для фильтрации (прореживания)
filter.routes.id[]	Название роута (должен совпадать с access_log.route_id)
filter.routes.id[].enabled	Включен ли фильтр для роута (по умолчанию true)
filter.routes.id[].limit.duration	Длительность временного окна или размер size (одна из опция обязательна) по стандарту ISO 8601 duration format . Пример: PT10S
filter.routes.id[].limit.capacity	Максимальное кол-во запросов
filter.routes.id[].criteria	Критерий уникальности запроса (по умолчанию routeld). Пример: method,url

Настройка алертов

application.alerts.targets – раздел определяет список действий при срабатывании события, например логирование через syslog . У каждого действия должно быть заполнено поле name и (опционально) поле properties для настройки.

application.alerts.rules – список правил для генерации алерта. Каждое правило состоит

из одного или нескольких условий и действия (target).

Обязательное условие для каждого алерта: route должен совпадать с названием route из конфигурации envoy-alc.

Дополнительное условие: url

Для выражений в поле url используется библиотека <https://commons.apache.org/proper/commons-jexl/> выражение фильтрации каждого запроса. Пример:

```
request.url == '/hi'
request.url =~ '/hi[0-9]+'
```

Доступные операции:

```
== - точное сравнение
 ^= - префиксное сравнение (для строк)
 =$ - суффиксное сравнение
 =~ - сравнение по регулярному выражению
 <,>,<=,>= - операции сравнения (для целых чисел или чисел с плавающей точкой)
```

Доступные поля:

- ⊕ request.url
- ⊕ request.headers
- ⊕ request.body
- ⊕ request.fromIp
- ⊕ request.fromPort
- ⊕ request.toHost
- ⊕ request.toPort
- ⊕ response.status
- ⊕ response.headers
- ⊕ response.body

Дополнительное условие: analyzers

Ограничить проверку статуса ответа анализатора только выбранным списком анализаторов. В этом случае условие status_condition срабатывает на указанный список, либо один из анализаторов (or) либо все (and).

Настройка Маскирования

В конфиг http-request-logger добавляется раздел application.mask с описанием маскирования на уровне всего приложения.

```
application
mask:
  class-analyzer: true
  enabled: true
  data-class:
    - CREDIT_CARD
    - FIO
    - PASSPORT
  rules:
    - in: response_payload,url,request_header,response_header,request_payload
      regex: some_\w+
```

```
- in: request_header
  header: "Authorization"
```

В разделе `mask.rules.header` указывается, содержимое какого заголовка должно маскироваться. Остальная часть запроса и ответа при этом сохраняется. Если в `mask.rules.in` указаны `request_header`, `response_header`, но не указан никакой конкретный заголовок и фильтр что-то нашел - замаскируются все заголовки. (Указанный заголовок маскируется не зависимо от выбранного типа данных и регулярного выражения) Параметр `mask.class-analyzer: true` указывает, надо ли использовать данные классификатора, `data-class`: указывает какие категории данных необходимо маскировать. Указанные категории, должны быть включены в классификаторе. Маскируется тело запроса и тело ответа, в деталях работы классификатора будут замаскированы все обнаруженные данные. Если включен параметр `class-analyzer: true` и не указаны классы в параметре `data-class`, то маскируются данные с любыми классами. Параметр `mask.rules.regex` указывает, что нужно использовать правила маскирования на основе регулярных выражений (Синтаксис аналогичен правилам сигнатурного анализатора). Во всех случаях маскируется тело запроса и тело ответа

Настройка Ограничения хранения событий

В конфиг `http-request-logger` добавляется раздел `application.filter` с описанием фильтрации на уровне всего приложения и отдельных роутов. Механизм реализован на основе временного окна.

Фильтр пропускает запрос если:

1. Статус проверки `checkStatus = FAIL`
2. Фильтр глобально отключен. отсутствует опция `defaultRoute` и `routes` либо `defaultRoute.enabled = false`
3. Правило пропускает запрос (кол-во запросов меньше `size` за период `duration`)
4. Если правило не задано для роута запроса, используется правило по-умолчанию, заданное в `defaultRoute`
5. Если указана опция `criteria`, то правило задается для каждого уникального варианта сочетания значений полей из запроса.

Раздел `inventory`

Параметр конфигурации	Назначение
<code>cidr-list</code>	Подсети, которые считаются внутренними. Подсети указываются в формате CIDR. Тип доступа определяется по IP адресу отправителя. Если в запросе есть заголовок <code>X-Forwarded-For</code> , то принадлежность к подсети приоритетно определяется по этому заголовку. Если адрес отправителя или значение заголовка <code>X-Forwarded-For</code> принадлежат подсети, то тип доступа будет <code>PRIVATE</code> . Если адрес или значение заголовка не принадлежит подсети, то тип доступа будет <code>PUBLIC</code> .
<code>sync.merge-threshold</code>	Определяет, после какого количества похожих эндпоинтов будет выполняться объединение по шаблону <code>STRING</code> . Если указано 5, то объединение произойдет после 6 запроса. Все запросы должны иметь в URL различие только в одном сегменте, параметры должны совпадать не менее, чем на 80

	%
http-status-codes	Решение инвентаризирует только те эндпоинты, к которым было хотя бы одно обращение с момента запуска решения. При этом по умолчанию система не инвентаризирует эндпоинты, у которых запросы завершились кодами групп 4XX и 5XX – они не появятся в списке эндпоинтов. Однако сами запросы будут зарегистрированы, и если какой-либо запрос к эндпоинту завершится кодом из группы 2XX или 3XX (например, первый запрос – 405, второй запрос – 201), то эндпоинт появится в списке эндпоинтов. Запросы с 4XX и 5XX не будут учтены при определении параметров запроса, но будут учтены при расчёте здоровья (отношение запросов с кодами 2XX и 3XX ко всем запросам).

Пример конфигурации:

```
inventory:
  cidr-list: 10.3.0.0/16, 192.169.1.0/24, 51.0.0.0/8, 151.249.0.0/16
  sync:
    merge-threshold: 5
  http-status-codes: 200-399
```

Раздел db

Партиционирование

По умолчанию запись логов осуществляется в патинированную таблицу. Через параметр **retention** возможно указать срок хранения данных (в месяцах). По умолчанию срок хранения установлен в три месяца.

Пример конфигурации

```
db:
  partition:
    retention: 3
```

ui

Модуль серверной части работы с пользовательским Web-интерфейсом.

Через переменные окружения в контейнер надо передать адрес модуля srv-request-logger.

Пример конфигурации:

```
environment:
  NGINX_SRV_BACKEND: "srv-request-logger:8100"
```

srv_config_server

Модуль управления конфигурацией системы

Через переменную окружения в контейнер модуля надо передать URL git-репозитория с конфигурацией и смонтировать путь к каталогу с ssh-ключом.

Пример конфигурации:

```
volumes:
  - type: bind
    source: /home/ci/.ssh/
    target: /root/.ssh/
environment:
  GIT_URL: "git@github.com:api-security-gateway/rules-config.git"
```

http-sniffer

Модуль зеркалирования, извлекает http-запросы и ответы из копии трафика и записывает их в kafka. После эти сообщения из kafka читает и обрабатывает envoy-alc.

http-sniffer запускается как сервис в ОС Linux:

```
sudo systemctl stop httpsnf
```

```
sudo systemctl start httpsnf
```

```
sudo systemctl status httpsnf
```

Пример конфигурации сервиса через /etc/systemd/system/httpsnf.service:

```
[Unit]
Description=Squad Http sniffer
After=network.target
StartLimitBurst=5
StartLimitIntervalSec=10

[Service]
Type=exec
Restart=on-failure
RestartSec=1
User=root
ExecStart=sudo /opt/http-sniffer/httpsnf -c /opt/http-sniffer/config.properties veth1

[Install]
WantedBy=multi-user.target
```

В ExecStart указывается путь к бинарному файлу httpsnf, путь к файлу конфигурации, а также имя интерфейса, на который подается копия трафика.

Основные параметры, которые задаются в файле config.properties:

Параметр конфигурации	Назначение
verbose=True	Расширенное логирование Пример: verbose=True
workers	Количество потоков (по-умолчанию: 2) Пример: workers=1
kafka-server	Сервер kafka и порт для отправки сообщений Пример: kafka-server=localhost:19092

kafka-topic	Наименования топика kafka для записи сообщений с http-запросами и ответами Пример: kafka-topic=accessLogForCheck
kafka-security kafka-user kafka-password	Протокол аутентификации (plaintext, sasl_plaintext, ssl, sasl_ssl) и учетные данные в kafka Пример: kafka-security=sasl_plaintext kafka-user=dev kafka-password=dev-password
log-file	Путь к лог-файлу Пример: log-file=/opt/http-sniffer/logs/http-sniffer.log
log-rotation-size log-files-count	Параметры ротации лога Пример: log-rotation-size=1000000 log-files-count=10
http-ports	Порты, для которых сниффер извлекает запросы и ответы Пример: http-ports=80,8080

Для того, чтобы узнать версию сниффера и дополнительные параметры можно использовать команду:

```
httpsnf -h
```

envoy_alc

Модуль интеграции с Envoy.

Раздел "application"

Параметр конфигурации	Назначение
access-log-port	Порт для получения accessLog от Envoy. Пример: 9011
config-refresh-rate-sec	Период обновления конфигурации. Пример: 30
check-timeout-sec	Максимальное время проверки запроса

	Пример: 5
request-queue-size	Размер внутренней очереди анализаторов
mode	Общий режим проверки Возможные значения: BLOCKED, TRANSPARENT
analyzers.id	Имя используемого анализатора
analyzers.mode	Режим проверки анализатора (необязательный параметр) Возможные значения: BLOCKED, TRANSPARENT Если не задан, то используется общий режим проверки (mode)
routes[].id	Название роута. Роут определяется для дальнейшего использования в других анализаторах Пример: demo-service-gw
routes[].mode	Режим проверки анализатора для этого роута (необязательный параметр) Возможные значения: BLOCKED, TRANSPARENT Если не задан, то используется общий режим проверки (mode)
routes[].envoy-config.host	Сетевой интерфейс envoy (если используется envoy)
routes[].envoy-config.port	Порт envoy (если используется envoy)
routes[].analyzers.id	Имя используемого анализатора (необязательный параметр)
routes[].analyzers.mode	Режим проверки анализатора (необязательный параметр) Возможные значения: BLOCKED, TRANSPARENT
mapper.ip.headers	HTTP-заголовки, из которых система будет определять реальный IP-адрес клиента (пр. X-Forwarded-For, X-Real-IP). (Поле "IP отправителя" на странице "События")

Наследование параметров

В конфигурации используется модель наследования параметров.

Режим проверки (mode) можно определить для всего модуля сразу. Он будет применяться во всех анализаторах всех роутов.

Анализаторы можно определить для всего модуля (analyzers). Для каждого анализатора можно задать режим проверки индивидуально. Если режим не задан, то будет

использоваться режим модуля.

Анализаторы можно определить для конкретного роута. Если анализаторы не заданы, то будут использованы анализаторы модуля.

Пример.

```
mode: BLOCKED
analyzers:
  - id: srv-swagger-analyzer
    mode: "BLOCKED"
routes:
  - id: "demo-service"
    mode: "TRANSPARENT"
    envoy-config:
      host: "0.0.0.0"
      port: 10000
    analyzers:
      - id: srv-signature-analyzer
        mode: "BLOCKED"
```

Для всего модуля задан режим работы BLOCKED.

Это значит, что если не задано иное, то все запросы будут обрабатываться в этом режиме.

Для всего модуля задан один анализатор srv-swagger-analyzer.

Это значит, что если не задано иное, то во всех запросах будет использован этот анализатор.

Для этого анализатора задан режим BLOCKED. Это значит, что для этого анализатора будет использован этот режим, режим уровня модуля будет проигнорирован.

Для роута задан режим TRANSPARENT.

Это значит, что режим заданный на уровне модуля, будет проигнорирован. И режим анализаторов, заданных на уровне модуля, тоже будет проигнорирован.

Для роута задан свой набор анализаторов (один srv-signature-analyzer).

Это значит, что будет использован только этот анализатор. Режим для анализатора задан, поэтому будет использован. Если бы режим не был задан, то использовался бы режим на уровне роута (TRANSPARENT).

Определение роута при использовании Envoy

Во входящем запросе анализируется host-port и ищется соответствие роуту из конфигурации. Если подходящий роут не найден, то используется роут по умолчанию.

Пример.

В конфигурационном файле envoy.yaml

```
listeners:
  - name: listener_0
    address:
      socket_address:
        address: 0.0.0.0
        port_value: 10000
```

Для этого listener-а в конфигурации envoy-als должно быть:

```
routes:
- id: "demo-service"
  envoy-config:
    host: "0.0.0.0"
    port: 10000
```

Обратите внимание на соответствие address = host и port_value = port

Определение роута при использовании Nginx

При использовании Nginx может быть применен механизм определения роута по порту, аналогичный envoy.

В дополнении к этому в конфиге Nginx можно указать имя роута.

Например:

```
on_request vampi host.docker.internal:9020 blocked;
```

В этом случае запросы из nginx в envoy-als придут с уже установленным роутом "vampi". Если в конфигурации envoy-als такого роута нет, то будет использован роут по умолчанию.

Определение роута при использовании http-sniffer

При использовании http-sniffer роуты определяются по URL.

В конфигурацию envoy-als добавляется application.kafka-routes в котором перечисляются роуты и их паттерны URL

Пример:

```
kafka-routes:
- id: "juice-shop"
  path-pattern: "^/juice/"
- id: "ecco"
  path-pattern: "^/ecco/"
- id: "demo"
  path-pattern: "^/response/"
- id: "root"
  path-pattern: "^/"
```

lua-скрипты

lua-скрипты для envoy и пример конфигурационного файла поставляются отдельным zip-архивом.

В скрипте api_properties.lua задается режим обработки запроса.

transparentMode = true – означает, что запросы будут отправляться в режиме TRANSPARENT.

transparentMode = false – означает, что запросы будут отправляться в режиме BLOCKED.

Настройка анализаторов

В состав модуля envoy-als входят анализаторы. Анализаторы настраиваются в этом же конфигурационном файле. Для включения анализатора используется параметр конфигурации enabled: true. С помощью параметра concurrency можно установить требуемое кол-во экземпляров анализатора.

srv-ml-gateway

Модуль-шлюз, обеспечивающий связь системы с анализаторами.

Конфигурация модуля (раздел ml-gateway).

Параметр	Назначение
concurrency	кол-во создаваемых экземпляров анализатора
check-request-order-no	Какой по счету запрос отправлять на проверку в ml модуль. Например check-request-order-no: 3 - отправлять каждый третий запрос check-request-order-no: 2 - отправлять каждый второй запрос check-request-order-no: 1 - отправлять все запросы на проверку
cache-duration-sec	Продолжительность хранения похожих запросов в кэше (сек)
exclude-headers	Заголовки, которые будут игнорироваться при принятии решения о кэшировании Пример: date, x-envoy-upstream-service-time

Конфигурация отдельных модулей.

Подраздел analyzer.

class-analyzer

Модуль анализатора чувствительных данных.

Для ограничения допустимых к обращению чувствительных данных нужно для каждого роута настроить допустимые категории. Если категории не назначены, то анализатор блокирует запросы, в которых обнаружена хотя бы одна категория. Пример конфигурации, где разрешены все категории:

```
analyzer:
  class:
    enabled: true
  routes:
    - id: "ecco"
      data-class: LOCATION, RU_BANK_ACCOUNT_NUMBER, CREDIT_CARD, EMAIL_ADDRESS, FIO, INN,
PASSPORT, PHONE_NUMBER, SNILS
```

Категории персональных данных (в логге дублируется на русском и английском):

- ⊕ "LOCATION", "Адрес, местоположение";
- ⊕ "RU_BANK_ACCOUNT_NUMBER", "Номер счета";
- ⊕ "CREDIT_CARD", "Номер кредитной карты";
- ⊕ "EMAIL_ADDRESS", "Email адрес";
- ⊕ "FIO", "ФИО";
- ⊕ "INN", "ИНН";
- ⊕ "PHONE_NUMBER", "Номер телефона";
- ⊕ "SNILS", "Снилс";
- ⊕ "PASSPORT", "Паспорт".

Добавление ключевых слов для определения серии и номера паспорта, номера снисла и

ИНН (на примере паспорта)

Параметр	Назначение
application.ml-gateway.passport.keywords-check-enabled	Включение/отключение проверки ключевых слов (по умолчанию: false)
application.ml-gateway.passport.keywords	Список ключевых слов для поиска перед номером паспорта (по умолчанию: пустой список)
application.ml-gateway.passport.max-keyword-distance	Максимальное расстояние в символах для поиска ключевых слов перед номером паспорта (по умолчанию: 10)

Режим поиска паспортов (PassportSearchMode)

Можно выбрать один или несколько режимов поиска паспортов. Если указано несколько режимов, поиск выполняется по каждому из них, результаты объединяются.

Параметр	Назначение
application.ml-gateway.passportsearch-modes:	
DIGITS_ONLY_WITH_SPACE_REMOVAL	Ищется 10 цифр подряд. Исходный текст очищается: пробелы между цифрами удаляются
DIGITS_ONLY_WITHOUT_SPACE_REMOVAL	Ищется 10 цифр подряд без разделителей. Пробелы в тексте сохраняются
SEPARATED_SERIES_AND_NUMBER_BY_SPACE	Серия (4 цифры) и номер (6 цифр) разделены пробелом (по умолчанию)
SEPARATED_INSIDE_SERIES_AND_NUMBER_BY_SPACE	Серия разделена внутри (2 + 2 цифры), номер отделён пробелом (6 цифр).

Удаляются полностью (могут быть в номере и в "пробеле" между номером и серией):
-()'"`

Заменяются на пробел (могут быть в "пробеле" между номером и серией):
.,;<>{ }

Кастомные классы чувствительных данных (custom-data-class) - создание и настройки пользовательских правил обнаружения чувствительных данных. Эти правила позволяют расширить возможности классификатора.

custom-data-class:

```
rules:
- data-class: class
  data-class-title: класс
  regex: class\d?
```

Описание полей:

- ⊕ data-class: class - внутреннее имя класса;
- ⊕ data-class-title - имя, которое будет отображаться в интерфейсе;
- ⊕ regex: class\d? - шаблон (regex), по которому анализатор будет искать совпадения в тексте запроса или ответа.

Созданный кастомный класс автоматически отображается во всех разделах системы, где выводится список классов чувствительных данных — например, в фильтрах, выпадающих списках и таблицах с результатами анализа.

srv_signature_analyzer

Модуль сигнатурного анализатора.

Раздел "signature-analyzer"

Параметр конфигурации	Назначение
concurrency	кол-во создаваемых экземпляров анализатора
rules.reload-period-sec	период вычитывания правил (сек)
all-rules-check: true	включает проверку всех правил (либо проверка остановиться при первом нарушенном правиле)

Сигнатурный анализатор содержит множество предустановленных правил для проверки поступающих запросов. Поддерживается возможность как включения/выключения правил по умолчанию, так и добавления пользовательских экземпляров.

Основной конфигурационный файл `_rule_status_list.json` должен находиться в папке `rules/config` относительно git-репозитория указанного в конфигурации.

Для включения/выключения правила:

- 1) Найти правило с нужным id по имени файла в `_rule_status_list.json`.
- 2) Перевести флаг `enabled` в требуемое состояние: `true` – включено, `false` – выключено.

Для добавления пользовательских правил:

- 1) Добавить json-файл с правилом в папку `rules` в git-репозитории.
- 2) Добавить в `_rule_status_list.json` конфигурационную запись в формате:

```
{
  "id": "custom_rule_filename",
  "enabled": true
}
```

где `id` – имя файла с пользовательским правилом.

Обновление набора правил и их конфигурации произойдет согласно установленному

периоду вычитывания правил или после перезапуска сигнатурного анализатора.

srv_swagger_analyzer

Модуль анализа на основе swagger-схем.

Раздел: "swagger-analyzer"

Параметр конфигурации	Назначение
concurrency	кол-во создаваемых экземпляров анализатора
schema.reload-period-sec	Период перечитывания swagger-схем (сек). Пример: 10
schema.routes[].id	Название роута. Роут определяется в модуле проксирования. Пример: demo-service-gw
schema.routes[].api-prefix	Префикс api-gateway. Параметр задается, если приложение находится за api-gateway. См. описание ниже.
schema.routes[].file	Имя файла схемы swagger Пример: DemoApplication.json

Префикс api-gateway.

Параметр задается, если приложение находится за api-gateway.

Пример: "demo-service".

В этом случае URL типа "/demo-service/api/test" будет анализироваться на соответствие схеме как урл: "/api/test" (Также нужно отредактировать server.url (+ prefix) в свагерр-схеме)

Несколько префиксов можно задать через регулярное выражение.

Например, если префикс определен так: "^/api/v\d/account"

то он будет срабатывать на:

"/api/v1/account"

"/api/v2/account"

...

"/api/v9/account"

если так: "^/api.*/account", то

будет работать с:

"/api/account/operation1"

"/api/v1/account/operation1"

Желательно регулярное выражение начинать с символа "^" (начало строки), но не обязательно.

такой префикс "/api.*/account"

сработает на

"/appl/api/v4/account/operation1"

Есть вероятность нескольких совпадений по префиксам.

Например, префикс `"/api/list"` формально сработает дважды на путь `"/api/list/operation1/api/list/operation"`
Если найдется больше одного совпадения, то считается, что совпадений не было.

Возможен и такой пример описания: `"^/api/(account|card|client)"`
для такого префикса подойдут пути:
`"/api/account/operation1"`,
`"/api/card/operation1"`,
`"/api/client/operation1"`

Пример работы с запросами:

Пусть prefix: `"/prefix"`. В сваггер схеме есть эндпоинт `/someUrl`.

Урл1: `/prefix/someUrl`. После обработки будет `/SomeUrl`. Такой эндпоинт есть, запрос точно прошёл проверку по существованию API path. Анализатор будет проверять запрос на соответствие другим параметрам, которые должны быть по схеме.

Урл2: `/word/someUrl`. После обработки останется `/word/someUrl`. Потому что там вообще нет нашей строки с префиксом. В итоге имеем эндпоинт `/word/someUrl`. Такого эндпоинта в схеме нет. Анализатор заблокирует запрос по причине `"ERROR - No API path found that matches request"`, потому что в схеме нет такого эндпоинта.

srv_statistic_analyzer

Модуль статистического анализатора.

Раздел `"statistic-analyzer"`

Параметр конфигурации	Назначение
<code>concurrency</code>	кол-во создаваемых экземпляров анализатора
<code>analyzer.rules[].id</code>	Название правила Пример: <code>"test1"</code>
<code>analyzer.rules[].route</code>	Название роута. Роут определяется в модуле проксирования. Пример: <code>test1</code>
<code>analyzer.rules[].filter</code>	Фильтр запросов Пример: <code>"request.url == '/hi'"</code>
<code>analyzer.rules[].count</code>	Кол-во запросов, при достижении которого запросы блокируются
<code>analyzer.rules[].duration</code>	Период счетчика запросов по стандарту ISO 8601 duration format . Пример: <code>PT10S</code>
<code>analyzer.rules[].group-mode</code>	Режим группировки запросов.

	Возможные значения: ALL, IP, USER
analyzer.user-id-header	Имя заголовка с идентификатором пользователя. Пример: "X-Session-Id"
application.statistic-analyzer.rules[].statistic-mode	Режим работы RELATIVE или ABSOLUTE
application.statistic-analyzer.rules[].relative-min-count	Порог количества запросов в текущем окне, когда срабатывает относительный порог
application.statistic-analyzer.rules[].relative-threshold-percent	Процент изменения количества запросов за период времени duration

Статистический анализатор. Анализирует сообщения в топике кафки httpRequest, отправляет лог анализа в топик requestCheckResultLog. Собирает статистику по запросам за указанный период времени, если кол-во запросов превышает указанное значение, все последующие запросы считаются как FAIL.

Режимы работы

Абсолютный - собирает статистику по запросам за указанный период времени, если кол-во запросов превышает указанное значение, все последующие запросы считаются как FAIL.

Относительный - отслеживает процентное изменение нагрузки между двумя временными окнами и срабатывает при превышении порога.

Режимы агрегации (группировки)

ALL (Все запросы) – Все запросы, поступающие на роут, участвуют в статистике

IP (По IP) – Запросы группируются по IP-адресу, в статистике участвуют запросы на каждый айпи адрес

USER (По идентификатору пользователя) – Запросы группируются по идентификатору пользователя, идентификатор пользователя может быть передан через Basic-auth заголовок, либо через JWT токен, либо через специальный заголовок, например, X-Session-Id

Анализатор считает запрос как правильный OK, если кол-во таких запросов (подпадающих под правило) меньше count за указанный промежуток времени duration.

Запросы подпадают под правило в том случае, если совпадает название роута route, и, если указан фильтр filter, успешно срабатывает условие фильтра.

Для режима группировки USER доступно получение идентификатора пользователя следующими способами:

1. Из заголовка Authorization, тип Basic
2. Из заголовка Authorization, тип Bearer (JWT)
3. Из специального заголовка, название которого указано в конфигурации в поле user-id-header: X-Session-Id

Настройка параметра: filter

Для выражений в фильтре используется библиотека <https://commons.apache.org/proper/commons-jexl/>. Пример

```
request.url == '/hi'
request.url =~ '/hi[0-9]+'
```

Доступные операции

== - точное сравнение
 ^= - префиксное сравнение (для строк)
 =\$ - суффиксное сравнение
 =~ - сравнение по регулярному выражению
 <,>,<=,>= - операции сравнения (для целых чисел или чисел с плавающей точкой)

Доступные поля

```
request.url
request.headers
request.body
request.fromIp
request.fromPort
request.toHost
request.toPort
```

Исключения анализаторов

Со страницы UI События-*"Событие из списка"*-Анализ запроса/ответа возможно добавить сработку правила одного из анализаторов в исключения.

Все добавленные исключения сгруппированы по роутам и отображаются в UI События-Исключения анализаторов (В этом разделе исключения также можно удалить)

Правила исключений добавляются в таблицу Postgres "analyzer_bypass"

Конфигурирование ролей и привилегий

Решение поддерживает возможность конфигурирования RBAC политики.

Аутентификация выполняется средствами LDAP, далее в процессе авторизации пользователю назначается роль и соответствующие привилегии.

Основные конфигурационные файлы _users_config.json и _roles_config.json должны находиться в папке rbac относительно git-репозитория, указанного в конфигурации.

Перед началом использования решения сконфигурируйте пользователя с предустановленной ролью Configurator.

Роль 'Configurator' включает привилегии для модификации конфигурации доступа пользователей в системе.

Роль 'Configurator' может быть назначена пользователю двумя способами:

1. Конфигурация свойства application.rbac.configurator сервиса srv-request-logger:

```
application:
  rbac:
    configurator: "usr"
```

2. Назначение роли пользователю в файле конфигурации git:

```
{
  "username": "usr",
  "roles": [
    "Configurator"
  ]
}
```

Каждому пользователю после входа в систему назначается базовая роль 'User'. Роль 'User' недоступна для удаления, но возможна конфигурация базовых привилегий роли.

Пример базовой конфигурации пользователей (_users_config.json):

```
{
  "users": [
    {
      "username": "usr",
      "roles": [
        "User",
        "Configurator"
      ]
    },
    {
      "username": "guest",
      "roles": [
        "User"
      ]
    }
  ]
}
```

username – имя пользователя, настроенное в LDAP для доступа в систему.

roles – перечень ролей, назначенных пользователю. Роли должны соответствовать ролям, указанным в конфигурации.

Пример конфигурации ролей (_roles_config.json):

```
{
  "roles": [
    {
      "role": "User",
      "privileges": ["READ_REQUEST_BODY", "READ_RESPONSE_BODY"]
    },
    {
      "role": "Configurator",
      "privileges": ["WRITE_RBAC", "READ_RBAC"]
    },
    {
      "role": "Empty",
      "privileges": []
    }
  ]
}
```

role – название роли.

privileges – перечень привилегий, соответствующих роли.

Список доступных в системе привилегий:

1. WRITE_RBAC – разрешается создавать/изменять роли через пользовательский интерфейс.
2. READ_RBAC – разрешается видеть роли в пользовательском интерфейсе.
3. READ_REQUEST_BODY – разрешается видеть тела запроса во вкладке События.
4. READ_RESPONSE_BODY – разрешается видеть тело ответа во вкладке События.
5. WRITE_PROPERTIES – разрешается изменение настроек приложений
6. READ_PROPERTIES – разрешается получение настроек приложений
7. MANAGE_ANALYZER_BYPASS - разрешается управление исключениями анализаторов

Отсутствие какой-либо привилегии является запретом на соответствующее действие. Разрешается создание пользовательских ролей с произвольным набором привилегий, существующих в системе. Удаление роли User запрещено политикой системы.

Мониторинг работы системы

Healthcheck

Для понимания, работает ли модуль, можно воспользоваться REST-запросом, который обрабатывает встроенный в модуль Web-сервер.

Порт встроенного web-сервера задается в параметре конфигурации "WEB_PORT" для следующих модулей:

- ⊕ `srv_ml_attack_analyzer`
- ⊕ `srv_ml_class_analyzer`

REST-запрос для проверки `/health`

Порт встроенного web-сервера задается в параметре конфигурации "server.port" для следующих модулей:

- ⊕ `srv_signature_analyzer`
- ⊕ `srv_statistic_analyzer`
- ⊕ `srv_swagger_analyzer`
- ⊕ `envoy_alc`
- ⊕ `srv_request_logger`
- ⊕ `srv_config_server`

REST-запрос для проверки `/actuator/health`

Создание ключа для git-репозитория

Для создания и применения ключа надо выполнить следующие действия.

1. Создание ssh-ключей
в каталоге `~/.ssh` (если его нет – создать) выполнить:

```
ssh-keygen -m PEM -t ecdsa -b 256
```

на все вопросы оставить дефолтные ответы.

2. Конфигурирование ssh-ключей

в каталоге ~/.ssh создать файл config с содержимым:

```
Host github.com
  IdentityFile ~/.ssh/id_ecdsa
```

3. Загрузка публичного ключа в git-репозиторий в каталоге ~/.ssh должны быть файлы:
 id_ecdsa – приватный ключ.
 id_ecdsa.pub – публичный ключ.
 публичный ключ надо загрузить в git-репозиторий.
 Пример для github: <https://github.com/settings/keys>

Настройка Envoy

Вместе с системой поставляется архив с конфигурационными файлами для интеграции с Envoy.

Список файлов:

- ✦ api_functions.lua – редактирование файла не предусматривается, содержит скрипты.
- ✦ api_properties.lua – базовые настройки.
- ✦ envoy.yaml – основной файл конфигурации.
- ✦ docker-compose.yaml – пример запуска envoy.

Возможно два варианта использования Envoy: работа с уже существующим инстансом и запуск нового инстанса специально для работы с системой. Если используется уже существующий инстанс, то в его конфигурацию надо добавить lua скрипты из поставки и внести изменения в конфигурационный файл по примеру envoy.yaml. Далее будет описано использование инстанса, созданного специально для работы системы.

Архив с конфигурационными файлами нужно разархивировать и выполнить следующие действия:

Внести в envoy.yaml изменения в соответствии с условиями эксплуатации.

В примере входящий трафик приходит в envoy на порт **10000** и перенаправляется на host.docker.internal, порты: **6060** и **6061**. В примере используется режим балансировки round robin.

Для проверки запросы направляются в сервис envoy-alc на порт **9010**, записи accessLog отправляются в сервис envoy-alc на порт **9011**. Эти параметры надо скорректировать.

1. В разделе static_resources.listeners надо изменить port_value (в примере **10000**) на требуемый.
 На этом порту Envoy будет принимать входящий трафик.
2. Убедиться, что в разделе static_resources.clusters (name: grpc_als_cluster).load_assignment.socket_address (значение в примере address: host.docker.internal, port_value: **9011**) указывается хост и порт, который слушается сервисом envoy-alc (параметр access-log-port в конфигурации envoy-alc)
3. Убедиться, что в разделе static_resources.clusters (name: backend_service).load_assignment.endpoints.lb_endpoints.endpoint.address прописаны хост и порт (в примере address: host.docker.internal, port_value: **6060**), на которые будет перенаправлен трафик. Если целевое приложение не поддерживает health checks,

то раздел `health_checks` надо удалить, иначе прописать в `http_health_check.path` соответствующий url (в примере `/actuator/health`). Если режим балансировки не требуется, то endpoint с портом 6061 можно удалить.

4. Убедиться, что в разделе `static_resources.clusters (name: checker_service)` `.load_assignment.endpoints.lb_endpoints.endpoint.address` прописаны хост и порт (в примере `address: host.docker.internal, port_value: 9010`), который слушает сервис `envoy-alc` (параметр `server.port`).

Внести изменения в файл `api_properties.lua`.

Если предполагается работа в режиме `TRANSPARENT` (т.е. все запросы будут проверяться, но не блокироваться), то параметр `transparentMode = false` надо изменить на `transparentMode = true`.

Если предполагается работа в режиме `BLOCKED` (т.е. все запросы будут проверяться, и, возможно, блокироваться), то изменять файл не требуется.

Для запуска инстанса Envoy можно использовать прилагаемый файл `docker-compose.yaml`. Файл `docker-compose.yaml` предполагает, что `api_functions.lua`, `api_properties.lua`, `envoy.yaml` будут находиться в этом же каталоге.

Настройка Nginx

Система поставляется с набором файлов интеграции с nginx.

Список файлов:

- ⊕ `docker-compose.yaml`
- ⊕ `nginx.conf`
- ⊕ `routing.conf`

`docker-compose.yaml` – это docker compose файл для быстрого запуска nginx в условиях пилота. Используется образ докера на основе типовой opensource версии nginx с включенным режимом `debug`.

`nginx.conf` – типовой конфигурационный файл nginx.

`routing.conf` – типовой конфигурационный файл nginx. В этом файле используется директива `qApiSec`.

В location, которые необходимо анализировать в qApiSec, нужно добавить директиву `on_request`.

Syntax: `on_request {имя роута}{location envoy-alc} blocked|nonblocked;`

Default: -

Context: `location`

Пример

```
location / {
    on_request vampi host.docker.internal:9020 blocked;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
```

```
proxy_set_header Host $http_host;
proxy_redirect off;
proxy_pass http://vampi;
}
```

Если envoy-als не доступен, то nginx пропустит запрос без проверки. Недоступность envoy-als не влияет на прохождение запросов.

Если в директиве используется значение blocked, то модуль envoy-als выполнит проверку запроса в соответствии с настройками роута. Запрос или ответ могут быть заблокированы, если в настройка роутов envoy-als есть соответствующая настройка.

Если в директиве используется значение nonblocked, то модуль envoy-als сразу же ответит модулю nginx - ОК. И такой запрос/ответ будут переданы дальше. Проверки в envoy-als все равно будут выполнены, но их результаты уже не повлияют на прохождение запроса.

Если требуется обеспечить максимальное быстроедействие и при этом допустимо игнорировать результаты проверки, то можно использовать режим nonblocked. Если результаты работы анализаторов необходимо учитывать, то нужен режим blocked.

Шифрование/дешифрование паролей

Вместе с системой возможна поставка сервиса шифрования/дешифрования паролей в файлах конфигурации jasypt-service и инструкции по его использованию.

Инструкция jasypt-service

Описание

Jasypt Service – это простой и надежный инструмент, который позволяет безопасно шифровать и дешифровать пароли. Используйте его для защиты конфиденциальных данных в файлах конфигурации.

Сервис поддерживает следующие параметры для запуска:

- ⊕ encrypt <plain_text> – зашифровать пароль
- ⊕ decrypt <encrypted_text> – расшифровать пароль

Используемый алгоритм шифрования:

- ⊕ PBEWITHHMACSHA512ANDAES_256

Запуск с помощью Docker Compose

Требования к среде выполнения:

- ⊕ docker-compose

docker-compose.yml:

```
services:
  jasypt-service:
    image: "cr.yandex/crpl8ec3g9qga1r9fui9/jasypt-service:latest"
    container_name: jasypt-service
```

```
command: ${ARGS:-encrypt dev}
environment:
  JASYPT_ENCRYPTOR_PASSWORD: ${MASTER_PASSWORD:-squadapisec}
```

1) Запуск приложения:

В каталоге с docker-compose.yml выполните команду:

```
ARGS="<command> <password>" MASTER_PASSWORD="<master_password>" docker-compose up
```

2) Запуск приложения с предустановленными аргументами:

Замените предустановленные значения переменных ARGS и MASTER_PASSWORD в файле docker-compose.yml на необходимые и выполните команду:

```
docker-compose up
```

Запуск с помощью Docker

Требования к среде выполнения:

 docker

Загрузка образа:

```
docker pull cr.yandex/crpl8ec3g9qga1r9fui9/jasypt-service:latest
```

Запуск приложения:

```
docker run --rm -e JASYPT_ENCRYPTOR_PASSWORD=<master_password> -it
cr.yandex/crpl8ec3g9qga1r9fui9/jasypt-service:latest <command> <password>
```

Использование

Пароли в зашифрованном виде указываются в файлах конфигурации сервисов, и далее сервис при старте автоматически дешифрует значение с использованием общедоступной библиотеки <http://www.jasypt.org/>.

Для осуществления функции шифрования паролей необходимо установить переменную среды JASYPT_ENCRYPTOR_PASSWORD, которая является мастер-паролем для шифрования либо дешифрования.

Пример использования:

1. Установить переменную JASYPT_ENCRYPTOR_PASSWORD в среде развертывания.
2. Использовать jasypt-service для шифрования пароля.
3. Заменить/вставить зашифрованный пароль ENC(...) в файл конфигурации сервиса, например:

```
application:
  kafka:
    sasl:
      username: "dev"
      password: ENC(P22jFiyDoalADWK6ZQYTaxt+eFbzZpE9zEdN/TVmFd7b4v/02NePX9yHzuK6tZfp)
```

Значение переменной JASYPT_ENCRYPTOR_PASSWORD и мастер-пароль используемый при шифровании пароля должны совпадать.

Работа в боевом режиме

Инфраструктурные компоненты

Системе для работы в боевом режиме требуются следующие инфраструктурные компоненты:

- ⊕ Kafka, версия 3.2.x и новее.
- ⊕ Postgres, версия 15 и новее.
- ⊕ Git репозиторий.
- ⊕ LDAP.
- ⊕ containers manager.

Используемые пароли очень желательно зашифровать (см. раздел "Шифрование/дешифрование паролей").

Kafka

Для подключения к Kafka потребуются url, имя пользователя и пароль.
В Kafka нужно создать следующие топики:

- ⊕ httpRequest.
- ⊕ accessLog.
- ⊕ requestToServiceLog.
- ⊕ requestCheckResultLog.

Кол-во партиций необходимо выбрать исходя из ожидаемой нагрузки. Если нагрузка не понятна, можно создать по четыре партиции.

Специфические параметры/расширения не требуются.

Postgres

Для подключения к Postgres потребуется url, имя пользователя и пароль.
У пользователя должны быть права на создание таблиц.
Необходимые таблицы будут созданы приложением при первом запуске.

Специфические параметры/расширения не требуются.

Git репозиторий

Для подключения к Git необходимо создать ключ (см. раздел "Создание ключа для git-репозитория").

LDAP

Для подключения к LDAP потребуются urls, base, username, password.
Пример:

```
urls: ldap://ldaphost:1389
base: ou=users,dc=example,dc=org,dc=users
username: cn=admin,dc=example,dc=org,dc=users
password: adminpassword
```

Containers manager

В качестве containers manager желательно использовать Kubernetes или его аналоги.

Ниже указаны контейнеры и ориентировочные требования по ресурсам.

- ⊕ srv-config-server (1 cpu, 512 Mb)
- ⊕ srv-request-logger (2 cpu, 3072 Mb)
- ⊕ ui (1 cpu, 256 Mb)
- ⊕ srv-ml-class-analyzer (2 cpu, 2048 Mb)
- ⊕ envoy-alc (4 cpu, 6144 Mb)

Интеграционное API

Назначение

Интеграционное API предназначено для передачи событий системы безопасности во внешние сервисы.

API предоставляет те же фильтры и атрибуты, что и интерфейс пользователя, и позволяет получать события в формате JSON.

Пример запроса:

```
``http://{URL}:8100/integration-api/v1/events?createdAtFrom=2026-01-22T00%3A00%3A00Z&createdAtTo=2026-01-22T23%3A50%3A50Z&limit=20&fromId=554479``
```

Базовый URL:

http://{URL}:8100/integration-api/v1/events

Обязательные параметры запроса:

Для выполнения запроса необходимо указать оба параметра временного диапазона:
?createdAtFrom=2026-01-22T00%3A00%3A00Z&createdAtTo=2026-01-22T23%3A50%3A50Z

Оба параметра являются обязательными.

При их отсутствии сервер возвращает ошибку:

400 Bad Request

```
{
  "errors": [
    {
      "message": "CreatedAtTo is required",
      "type": "ERROR",
      "header": "createdAtTo",
      "code": null
    }
  ]
}
```

Параметр fromId

Используется для страничной загрузки событий.

При указании fromId= API возвращает события, начиная со следующего идентификатора.

Параметр limit (не является обязательным)

Используется для ограничения выборки

Минимальное количество событий, возвращаемых API за один запрос- 25.

Если указать limit меньше 25, сервер вернёт ошибку:

400 Bad Request

"message": "limit must be greater or equal 25"

Пример:

GET /integration-api/v1/events?createdAtFrom=2026-01-22T00:00:00Z&createdAtTo=2026-01-22T23:50:50Z&limit=10

Ответ: 400 Bad Request

Корректный пример:

GET /integration-api/v1/events?createdAtFrom=2026-01-22T00:00:00Z&createdAtTo=2026-01-22T23:50:50Z&limit=25

Возвращается 25 событий, начиная с fromId (если указан).

Пример:

fromId=554481

В ответе будут события, начиная с eventId=554482.

Авторизация

Доступ осуществляется по API-ключу, передаваемому в заголовке:

X-API-Key: <ключ>

Пример тестовых ключей:

integration-api-key-456

test-api-key-123

При отсутствии или неверном ключе возвращается:

400 Bad Request

Примеры валидных запросов

Получение событий за период

GET http://{URL}:8100/integration-api/v1/events?createdAtFrom=2026-01-22T00%3A00%3A00Z&createdAtTo=2026-01-22T23%3A50%3A50Z&limit=25&fromId=554479

Ответ: JSON

```
{
  "total": 141,
  "items": [
    {
      "eventId": 554480,
```

```

"method": "GET",
"url": "/",
"fromIp": "64.62.156.10",
"fromPort": 58754,
"routeId": "demo-service",
"toHost": null,
"toPort": null,
"responseStatusCode": null,
"requestTime": "2026-01-22T01:58:07.708Z",
"createdAt": "2026-01-22T01:58:07.816Z",
"requestCheckStatus": "OK",
"responseCheckStatus": "OK",
"requestBlocked": false,
"responseBlocked": false,
"incomeCheckId": "42b56acd-82da-4368-b9d9-0c98da99feda",
"outcomeCheckId": null,
"requestBody": null,
"requestHeaders": {
  "Host": "89.169.137.241:10000",
  "Accept": "*/*",
  "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:56.0) Gecko/20100101 Firefox/56.0"
},
"responseBody": null,
"responseHeaders": {},
"blockedAnalyzers": [
  "swagger-analyzer"
],
"latitude": "44.9799690246582",
"longitude": "-93.26383972167969",
"proxyService": null,
"requestAnalyzers": [
  {
    "checkIdIncome": "42b56acd-82da-4368-b9d9-0c98da99feda",
    "checkIdOutcome": null,
    "checkStatus": "OK",
    "meta": null,
    "analyzerName": "statistic-analyzer",
    "date": "2026-01-22T01:58:07.000Z"
  },
  {
    "checkIdIncome": "42b56acd-82da-4368-b9d9-0c98da99feda",
    "checkIdOutcome": null,
    "checkStatus": "OK",
    "meta": {
      "type": "signature",
      "signatureDescription": null,
      "signatureFilename": null,
      "signatureDetect": [],
      "result": []
    },
    "analyzerName": "signature-analyzer",
    "date": "2026-01-22T01:58:07.000Z"
  },

```

```
{
  "checkIdIncome": "42b56acd-82da-4368-b9d9-0c98da99feda",
  "checkIdOutcome": null,
  "checkStatus": "OK",
  "meta": {
    "type": "ml_class",
    "result": []
  },
  "analyzerName": "class-analyzer",
  "date": "2026-01-22T01:58:07.000Z"
},
{
  "checkIdIncome": "42b56acd-82da-4368-b9d9-0c98da99feda",
  "checkIdOutcome": null,
  "checkStatus": "FAIL",
  "meta": {
    "type": "swagger",
    "schemaName": "Test Schema",
    "schemaFile": "demo-service-schema.json",
    "messages": [
      "ERROR - No API path found that matches request '/': []"
    ]
  },
  "analyzerName": "swagger-analyzer",
  "date": "2026-01-22T01:58:07.000Z"
}
],
"responseAnalyzers": []
}
]
```

Коды ответов

200 OK - Запрос выполнен успешно.

400 Bad Request - Ошибка параметров (например, `limit < 25`, неверный формат даты).

401 Unauthorized - Неверный или отсутствующий API-ключ.

500 Internal Server Error - Неподдерживаемый метод (например, `POST`).

Особенности реализации

- Минимальное значение параметра `limit`=25.
- События сортируются по возрастанию `createdAt`.
- Поддерживается пагинация по `fromId`.
- Возвращаются все поля, включая `requestAnalyzers`, `responseAnalyzers` и тела запросов/ответов.