?????????? KICS for Networks

- API KICS for Networks
- Настройка передачи событий из KICS for Networks (версий 4.3 и 4.5) в KUMA

API KICS for Networks

?????????????? Kaspersky Industrial CyberSecurity for Networks API

В Kaspersky Industrial CyberSecurity for Networks реализован интерфейс прикладного программирования, который обеспечивает доступ к функциям программы для сторонних приложений.

В комплект поставки Kaspersky Industrial CyberSecurity for Networks входит пакет с описаниями спецификаций для представления данных в запросах к серверу REST API. Сервер REST API функционирует на компьютере Сервера Kaspersky Industrial CyberSecurity for Networks и обрабатывает запросы с использованием архитектурного стиля взаимодействия REST. Обращения к серверу REST API выполняются по протоколу HTTPS. Вы можете настроить параметры сервера REST API в разделе Параметры → Серверы подключений (в том числе заменить используемый по умолчанию самоподписанный сертификат на доверенный).

Для представления данных в запросах и ответах используется формат JSON.

Документация с описанием запросов на основе архитектурного стиля REST публикуется в виде онлайн-справки на странице Kaspersky Online Help.

1. ОТКРЫТЬ ДОКУМЕНТАЦИЮ С ОПИСАНИЕМ ЗАПРОСОВ К СЕРВЕРУ REST API, версия 3
2. ОТКРЫТЬ ДОКУМЕНТАЦИЮ С ОПИСАНИЕМ ЗАПРОСОВ К СЕРВЕРУ REST API, версия 4

??????? ??????????????

Рассмотрим пример использования API для локального экспорта данных об устройствах из KICS for Networks.

Для начала нужно создать коннектор, с типом Generic и именем, например, **GetDeviceAll**, указать пароль и адрес сервера, узел размещения коннектора в нашем случае будет такой же, как и адрес сервера (Рисунок 1). После создания коннектора у вас скачается файл свертки, который будет называться как коннектор_GetDeviceAll.zip.

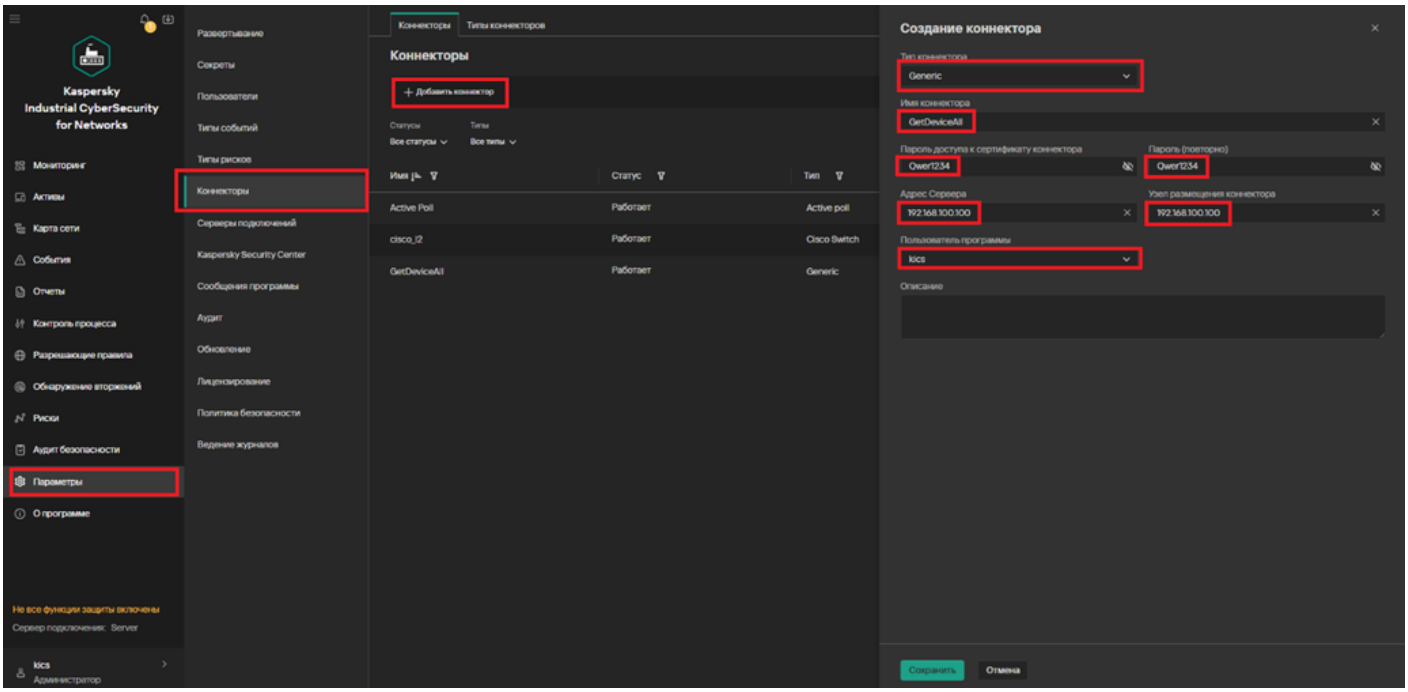


Рисунок 1 – Создание коннектора

Далее необходимо создать директории для корректной работы кода. Впишите команды ниже последовательно:

```
cd

mkdir GetDeviceAll && cd GetDeviceAll

mkdir connector && cd connector && cp /opt/kaspersky/kics4net-
connectors/libexec/default_connectors_registrar.py default_connectors_registrar.py
```

Далее в папку connector необходимо положить файл свертки коннектор_GetDeviceAll.zip, который мы получили при создании коннектора и распаковать его в эту директорию командой:

```
unzip -d коннектор_GetDeviceAll коннектор_GetDeviceAll.zip
```

Далее необходимо зарегистрировать коннектор командой:

```
sudo python3 default_connectors_registrar.py create
```

Далее необходимо выполнить действия показанные на Рисунке 2: ввести имя коннектора GetDeviceAll, ввести полный путь до файла свертки коннектор_GetDeviceAll.zip (/home/<username>/GetDeviceAll/connector/коннектор_GetDeviceAll.zip), ввести пароль, который вы указывали при создании коннектора в веб-интерфейсе и согласиться с запуском службы коннектора GetDeviceAll.

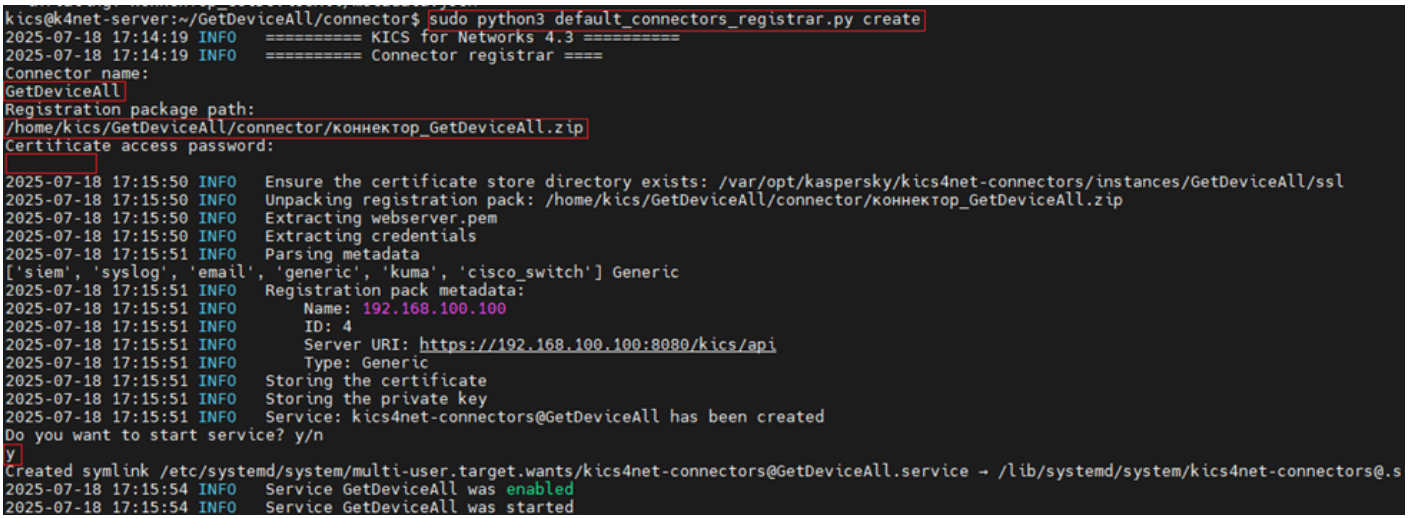


Рисунок 2 – Регистрация коннектора

После этого статус коннектора в KICS for Networks поменяется с “Ожидает регистрации” на “Работает”

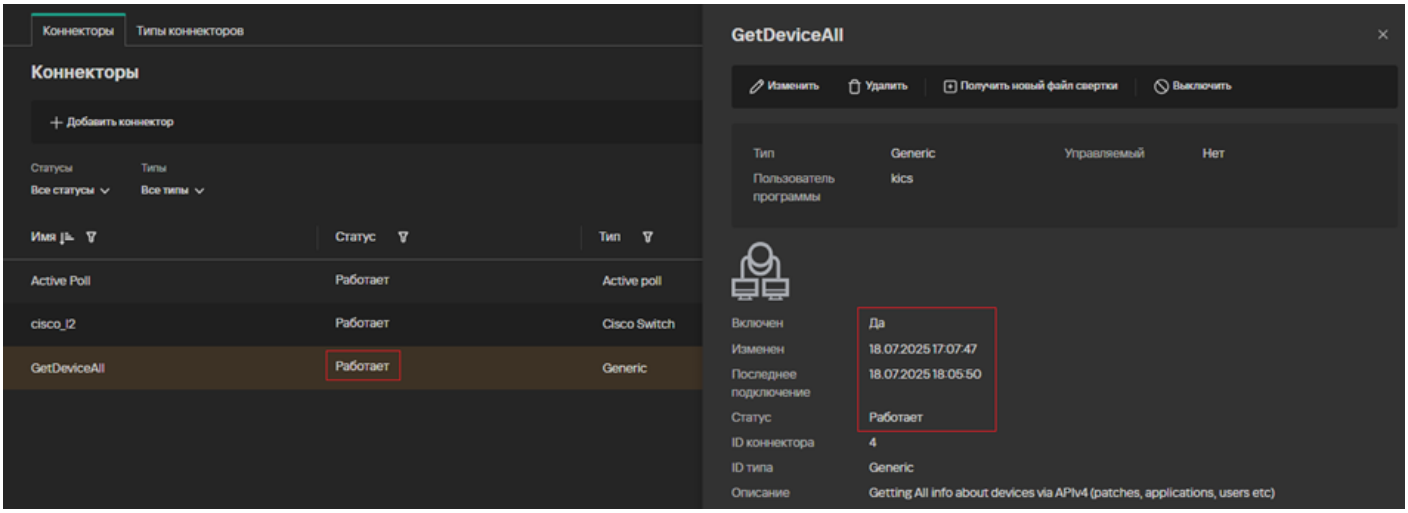


Рисунок 3 – Статус работы коннектора в Веб-консоли

Теперь можно перейти к написанию кода, который будет взаимодействовать с KICS for Networks по API.

> Код будет писаться на Python, но вы можете использовать любой удобный для вас язык

Для начала напишем класс подключения к API сервера:

```
import logging
import json
import requests
from requests.exceptions import RequestException
```

```
from time import sleep


class KicsApiClient:
    def __init__(self, metadata_path, cert_path, key_path):
        self.metadata_path = metadata_path
        self.cert_path = cert_path
        self.key_path = key_path
        self.base_url = None
        self.token = None
        self._initialize()

    def _initialize(self):
        with open(self.metadata_path, 'r', encoding='utf-8') as f:
            metadata = json.load(f)
        self.base_url = metadata.get('serverUri', '').rstrip('/')
        if not self.base_url.endswith('/kics/api'):
            self.base_url += '/kics/api'
        logging.info(f"Базовый URL API: {self.base_url}")
        self._authenticate()

    def _authenticate(self):
        url = f"{self.base_url}/auth/token?grant_type=certificate"
        resp = requests.post(
            url,
            cert=(self.cert_path, self.key_path),
            verify=False
        )
        resp.raise_for_status()
        self.token = resp.json().get('access_token')
        if not self.token:
            raise ValueError("Аутентификация не удалась: токен не получен")
        logging.info("Аутентификация успешна")

    def _post_query(self, path, payload, offset=0, limit=1000, delay=1):
        all_items = []
        headers = {
            'Authorization': f'Bearer {self.token}',
            'Content-Type': 'application/json'
        }
```

```
        while True:
            body = payload.copy()
            body.update({'offset': offset, 'limit': limit})
            url = f"{self.base_url}{path}"
            try:
                resp = requests.post(url, headers=headers, json=body, verify=False)
                resp.raise_for_status()
            except RequestException as e:
                logging.error(f"Ошибка запроса к {url}: {e}")
                if hasattr(e, 'response') and e.response is not None:
                    try:
                        logging.error(f"Текст ответа: {e.response.text}")
                    except:
                        pass
                raise
            try:
                json_resp = resp.json()
            except ValueError:
                break
            data = json_resp.get('values') or json_resp.get('elements')
            if not data:
                break
            all_items.extend(data)
            logging.info(f"Получено {len(data)} записей из {path}, всего: {len(all_items)}")
            if len(data) < limit:
                break
            offset += limit
            sleep(delay)
        return all_items
```

Теперь добавим в этот класс функции которые будут вызываться в классе экспорта данных:

```
    def get_device_applications(self):
        return self._post_query('/v4/device-apps/query', {})

    def get_device_patches(self):
        return self._post_query('/v4/device-patches/query', {})

    def get_device_users(self):
        return self._post_query('/v4/device-users/query', {})
```

Перейдем к написанию класса, который будет отвечать за экспорт данных из KICS for Networks по API и будет конвертировать полученные данные в csv файл:

```
import csv



class CsvExporter:
    @staticmethod
    def export_dicts(items, filename):
        if not items:
            logging.warning(f"Нет данных для файла {filename}")
            return
        fieldnames = set()
        for item in items:
            fieldnames.update(item.keys())
        fieldnames = sorted(fieldnames)
        headers_rus = [RUSSIAN_HEADERS.get(fn, fn) for fn in fieldnames]
        with open(filename, 'w', newline='', encoding='utf-8-sig') as f:
            writer = csv.writer(f, delimiter=';')
            writer.writerow(headers_rus)
            for item in items:
                row = [item.get(fn, '') for fn in fieldnames]
                writer.writerow(row)
        logging.info(f"Экспортировано {len(items)} записей в {filename}")
```

В этом классе используется глобальная переменная RUSSIAN_HEADERS, которая хранит в себе ключ - значение, для перевода ключей на русский язык:

```
RUSSIAN_HEADERS = {
    'Id': 'Идентификатор',
    'id': 'Идентификатор',
    'DeviceId': 'Идентификатор устройства',
    'device': 'Устройство',
    'ApplicationName': 'Название приложения',
    'name': 'Имя',
    'fullName': 'Полное имя',
    'ApplicationVersion': 'Версия приложения',
    'PatchVersion': 'Версия патча',
    'UserName': 'Имя пользователя',
    'description': 'Описание',
```

```
    'Domain': 'Домен',
    'groups': 'Группы',
    'isDisabled': 'Отключен',
    'isLockedOut': 'Заблокирован',
    'lastSeen': 'Последний вход',
    'canChangePassword': 'Можно сменить пароль',
    'mustChangePassword': 'Требуется смена пароля',
    'passwordNeverExpires': 'Пароль не истекает',
    'sid': 'SID',
    'source': 'Источник',
    'installedOn': 'Установлено на',
    'version': 'Версия',
    'vendor': 'Производитель',
    'program': 'Программа',
    'size': 'Размер',
}
```

Почти все готово! Осталось написать main, который объединит в себе весь функционал нашего кода:

```
import os
import sys



def main():
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    META = os.path.join(BASE_DIR, 'connector', 'коннектор_GetDeviceAll', 'metadata.json')
    CERT = '/var/opt/kaspersky/kics4net-connectors/instances/GetDeviceAll/ssl/connector.crt'
    KEY = '/var/opt/kaspersky/kics4net-connectors/instances/GetDeviceAll/ssl/connector.key'
    for path in (META, CERT, KEY):
        if not os.path.exists(path):
            logging.error(f"Не найден файл: {path}")
            sys.exit(1)
    client = KicsApiClient(META, CERT, KEY)
    apps = client.get_device_applications()
    patches = client.get_device_patches()
    users = client.get_device_users()
    CsvExporter.export_dicts(apps, 'DeviceApplications.csv')
    CsvExporter.export_dicts(patches, 'DevicePatches.csv')
    CsvExporter.export_dicts(users, 'DeviceUsers.csv')
```

```
if __name__ == '__main__':
    main()
```

Отлично! Код готов, теперь необходимо перенести файл GetDeviceAll.py в папку GetDeviceAll любым известным вам способом. В конечном счете директория должна выглядеть вот так:

```
└./home/<username>
       ├─ ./GetDeviceAll
       │  ├─ ./connector
       │  │  ├─ . /коннектор_GetDeviceAll.zip
       │  │  ├─ ./default_connectors_registrar.py
       │  │  └─ ./коннектор_GetDeviceAll
       │  │         ├─ ./certificates.pfx
       │  │         ├─ ./metadata.json
       │  │         └─ ./webserver.pem
       └  └─ ./GetDeviceAll.py
```

Далее в консоли перейдем в директорию GetDeviceAll командой:

```
cd /home/<username>
```

И запустить файл GetDeviceAll.py командой:

```
sudo python3 GetDeviceAll.py
```

В итоге мы получили 3 файла DeviceApplications.csv, DevicePatches.csv, DeviceUsers.csv которые находятся в этой же директории GetDeviceAll.

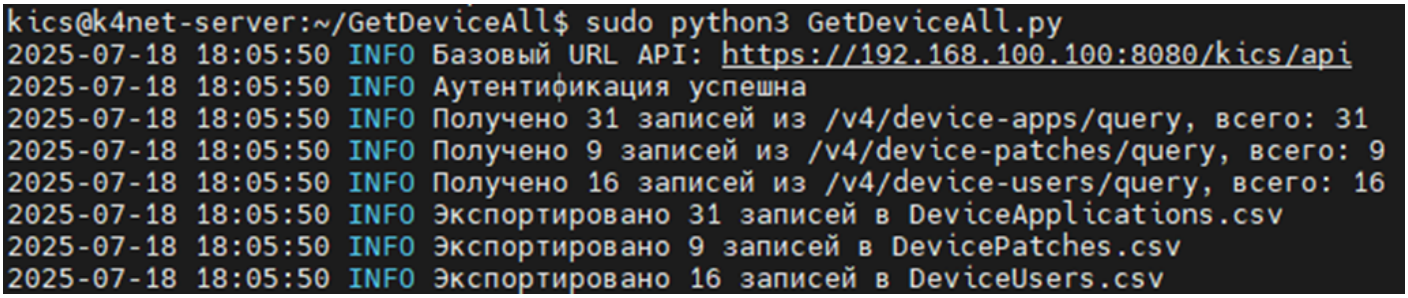


Рисунок 4 – Вывод логов скрипта по экспорту данных

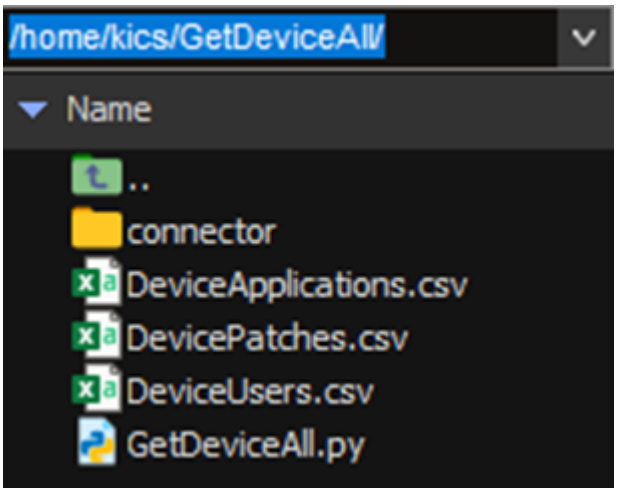


Рисунок 5 – Результат работы скрипта

P.S.

Полный код :)

```
import os
import sys
import json
import logging
import requests
import csv
from requests.exceptions import RequestException
import urllib3
from time import sleep

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

logging.basicConfig(
    format='%(asctime)s %(levelname)s %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    level=logging.INFO
)

RUSSIAN_HEADERS = {
    'Id': 'Идентификатор',
    'id': 'Идентификатор',
    'DeviceId': 'Идентификатор устройства',
    'device': 'Устройство',
    'ApplicationName': 'Название приложения',
```

```
    'name': 'Имя',
    'fullName': 'Полное имя',
    'ApplicationVersion': 'Версия приложения',
    'PatchVersion': 'Версия патча',
    'UserName': 'Имя пользователя',
    'description': 'Описание',
    'Domain': 'Домен',
    'groups': 'Группы',
    'isDisabled': 'Отключен',
    'isLockedOut': 'Заблокирован',
    'lastSeen': 'Последний вход',
    'canChangePassword': 'Можно сменить пароль',
    'mustChangePassword': 'Требуется смена пароля',
    'passwordNeverExpires': 'Пароль не истекает',
    'sid': 'SID',
    'source': 'Источник',
    'installedOn': 'Установлено на',
    'version': 'Версия',
    'vendor': 'Производитель',
    'program': 'Программа',
    'size': 'Размер',
}

class KicsApiClient:
    def __init__(self, metadata_path, cert_path, key_path):
        self.metadata_path = metadata_path
        self.cert_path = cert_path
        self.key_path = key_path
        self.base_url = None
        self.token = None
        self._initialize()

    def _initialize(self):
        with open(self.metadata_path, 'r', encoding='utf-8') as f:
            metadata = json.load(f)
        self.base_url = metadata.get('serverUri', '').rstrip('/')
        if not self.base_url.endswith('/kics/api'):
            self.base_url += '/kics/api'
        logging.info(f"Базовый URL API: {self.base_url}")
        self._authenticate()
```

```
def _authenticate(self):
    url = f"{self.base_url}/auth/token?grant_type=certificate"
    resp = requests.post(
        url,
        cert=(self.cert_path, self.key_path),
        verify=False
    )
    resp.raise_for_status()
    self.token = resp.json().get('access_token')
    if not self.token:
        raise ValueError("Аутентификация не удалась: токен не получен")
    logging.info("Аутентификация успешна")

def _post_query(self, path, payload, offset=0, limit=1000, delay=1):
    all_items = []
    headers = {
        'Authorization': f'Bearer {self.token}',
        'Content-Type': 'application/json'
    }
    while True:
        body = payload.copy()
        body.update({'offset': offset, 'limit': limit})
        url = f"{self.base_url}{path}"
        try:
            resp = requests.post(url, headers=headers, json=body, verify=False)
            resp.raise_for_status()
        except RequestException as e:
            logging.error(f"Ошибка запроса к {url}: {e}")
            if hasattr(e, 'response') and e.response is not None:
                try:
                    logging.error(f"Текст ответа: {e.response.text}")
                except:
                    pass
            raise
        try:
            json_resp = resp.json()
        except ValueError:
            break
        data = json_resp.get('values') or json_resp.get('elements')
```

```
            if not data:
                break
            all_items.extend(data)
            logging.info(f"Получено {len(data)} записей из {path}, всего: {len(all_items)}")
            if len(data) < limit:
                break
            offset += limit
            sleep(delay)
        return all_items

    def get_device_applications(self):
        return self._post_query('/v4/device-apps/query', {})

    def get_device_patches(self):
        return self._post_query('/v4/device-patches/query', {})

    def get_device_users(self):
        return self._post_query('/v4/device-users/query', {})

class CsvExporter:
    @staticmethod
    def export_dicts(items, filename):
        if not items:
            logging.warning(f"Нет данных для файла {filename}")
            return
        fieldnames = set()
        for item in items:
            fieldnames.update(item.keys())
        fieldnames = sorted(fieldnames)
        headers_rus = [RUSSIAN_HEADERS.get(fn, fn) for fn in fieldnames]
        with open(filename, 'w', newline='', encoding='utf-8-sig') as f:
            writer = csv.writer(f, delimiter=';')
            writer.writerow(headers_rus)
            for item in items:
                row = [item.get(fn, '') for fn in fieldnames]
                writer.writerow(row)
        logging.info(f"Экспортировано {len(items)} записей в {filename}")


def main():
```

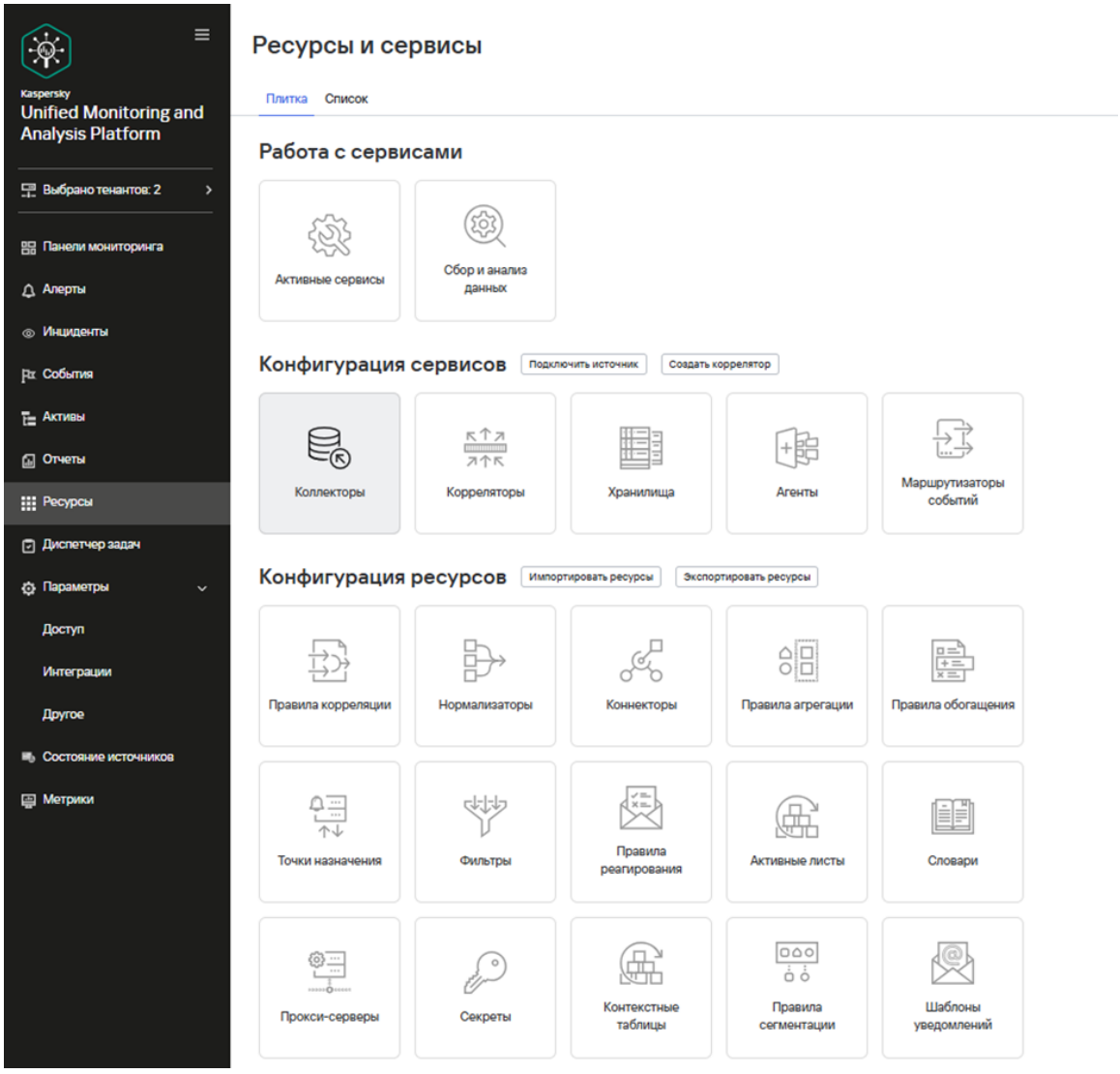
```
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    META = os.path.join(BASE_DIR, 'connector', 'коннектор_GetDeviceAll', 'metadata.json')
    CERT = '/var/opt/kaspersky/kics4net-connectors/instances/GetDeviceAll/ssl/connector.crt'
    KEY = '/var/opt/kaspersky/kics4net-connectors/instances/GetDeviceAll/ssl/connector.key'
    for path in (META, CERT, KEY):
        if not os.path.exists(path):
            logging.error(f"Не найден файл: {path}")
            sys.exit(1)
    client = KicsApiClient(META, CERT, KEY)
    apps = client.get_device_applications()
    patches = client.get_device_patches()
    users = client.get_device_users()
    CsvExporter.export_dicts(apps, 'DeviceApplications.csv')
    CsvExporter.export_dicts(patches, 'DevicePatches.csv')
    CsvExporter.export_dicts(users, 'DeviceUsers.csv')

if __name__ == '__main__':
    main()
```

????????? ???????? ??????? ?? KICS for Networks (?????? 4.3 ? 4.5) ? KUMA

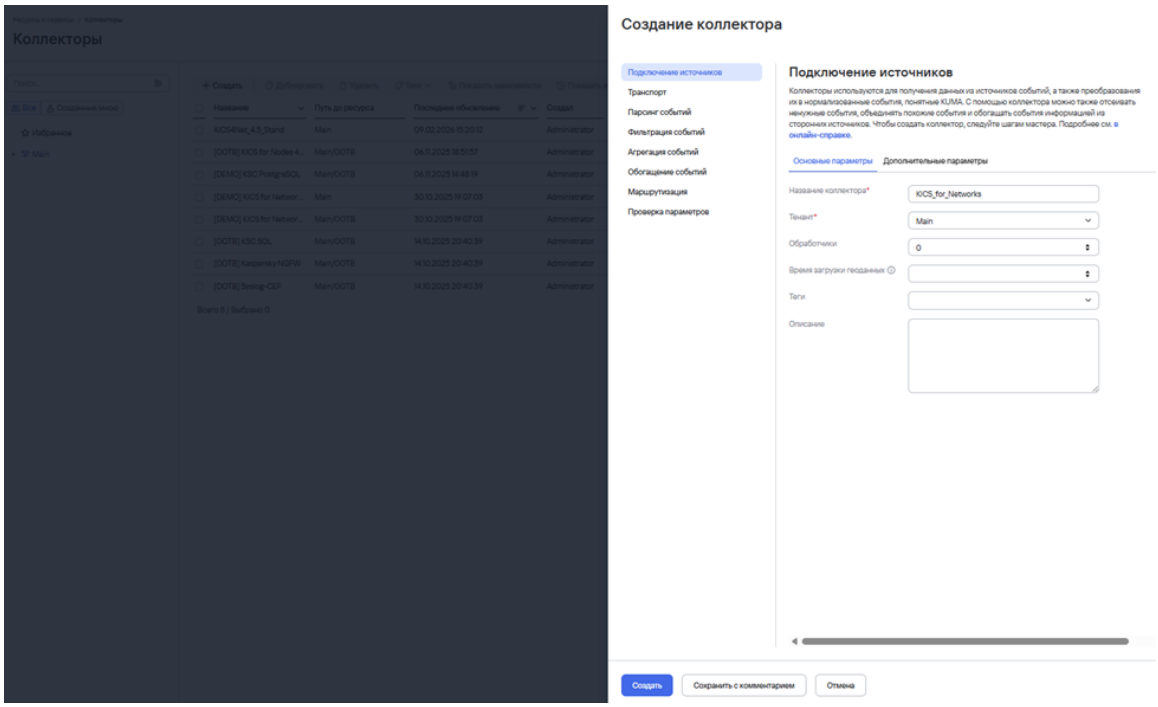
Настройка интеграции выполняется в два этапа: сначала создается и настраивается коллектор в KUMA, затем создается коннектор в KICS for Networks.

Настройка KUMA

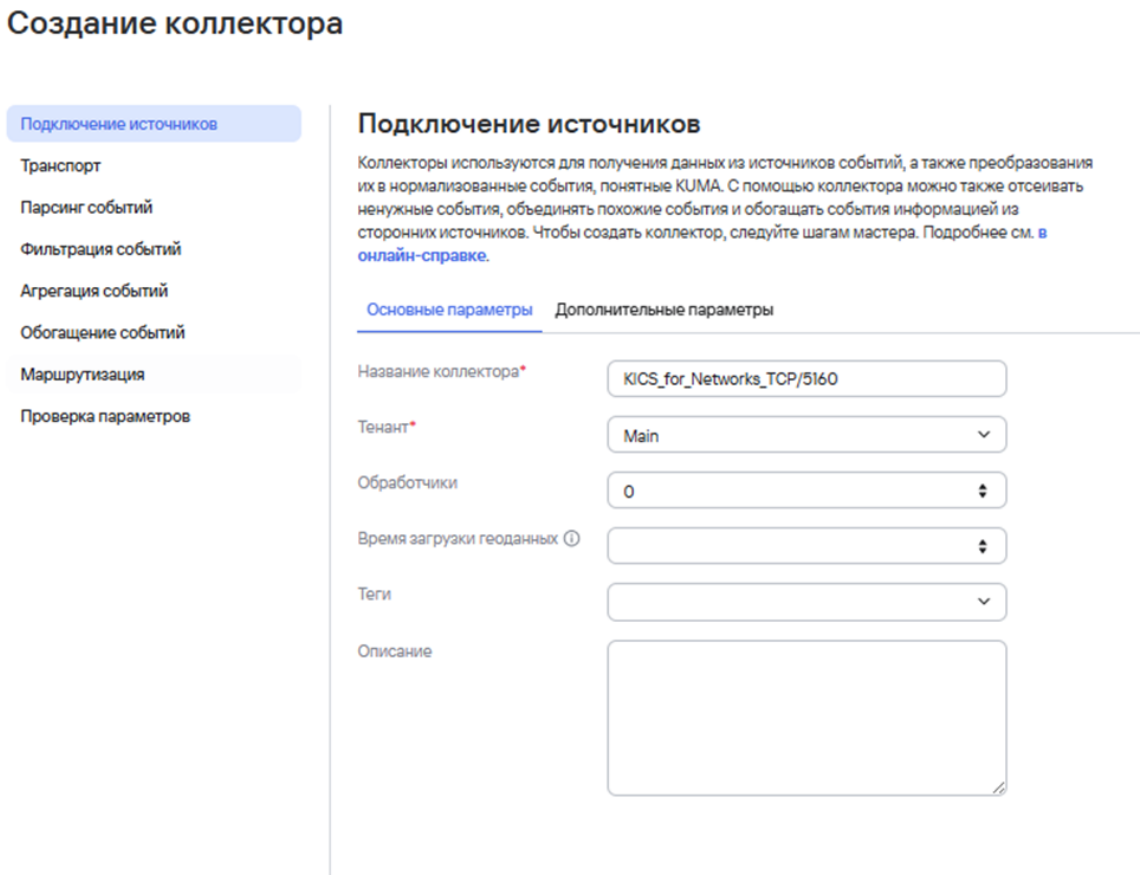
1. Перейдите в раздел «Ресурсы» и откройте вкладку «Коллекторы».



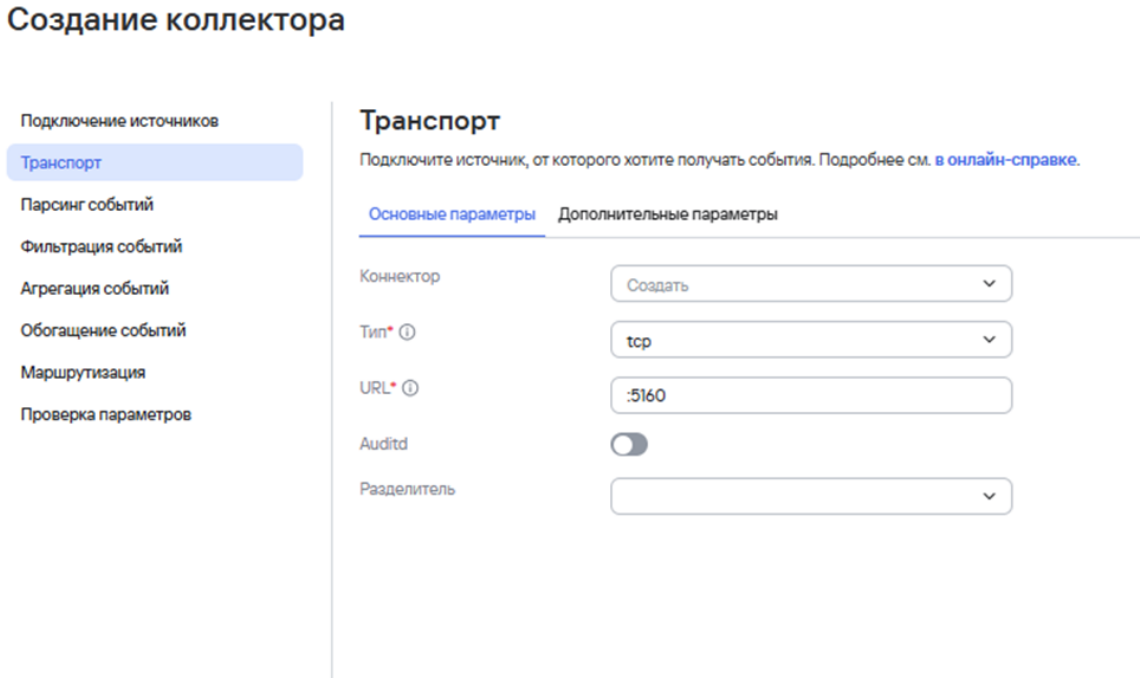
1. Нажмите кнопку «+ Создать».



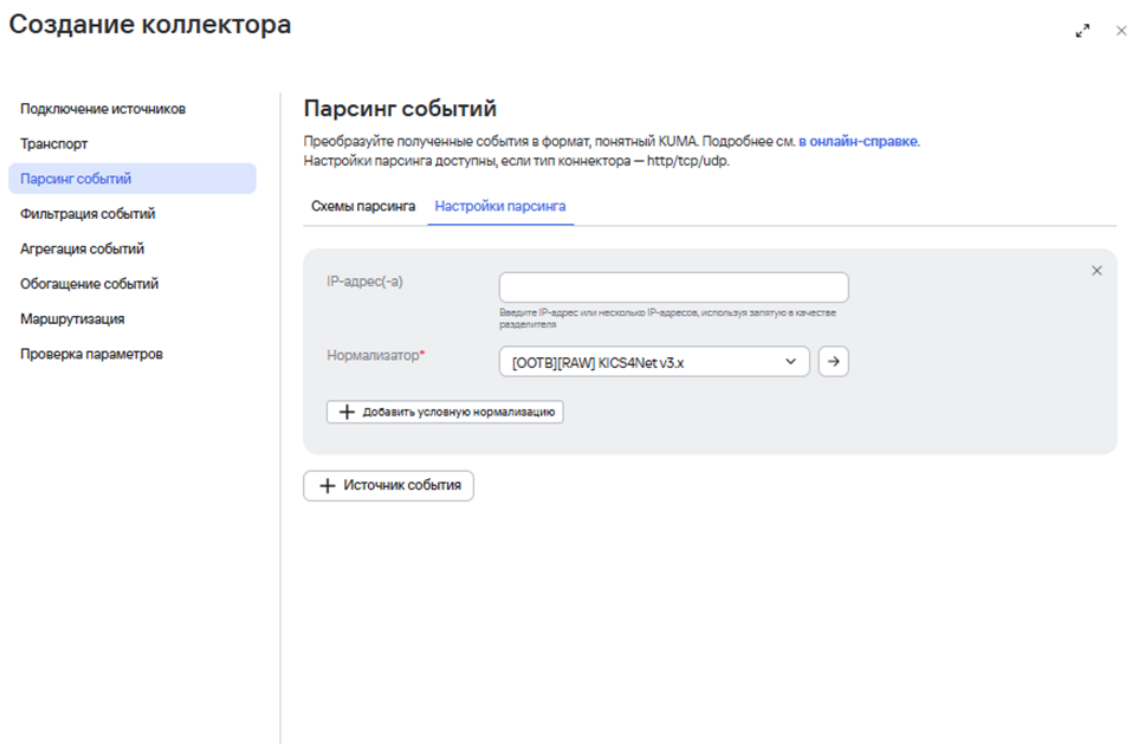
1. В открывшемся окне укажите название коллектора. Рекомендуется использовать наименование продукта, тип подключения и порт (например, KICS_for_Networks_TCP\5160), чтобы в дальнейшем легко идентифицировать коллектор.



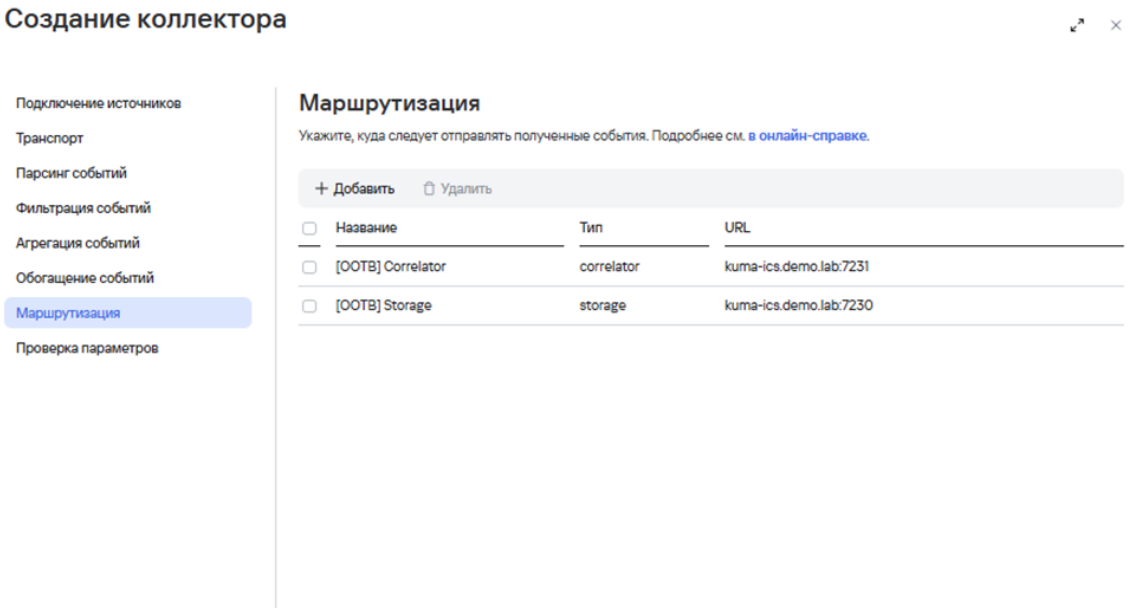
1. На вкладке «Транспорт» укажите тип соединения (протокол) и порт, который будет использоваться для приема событий от источника.



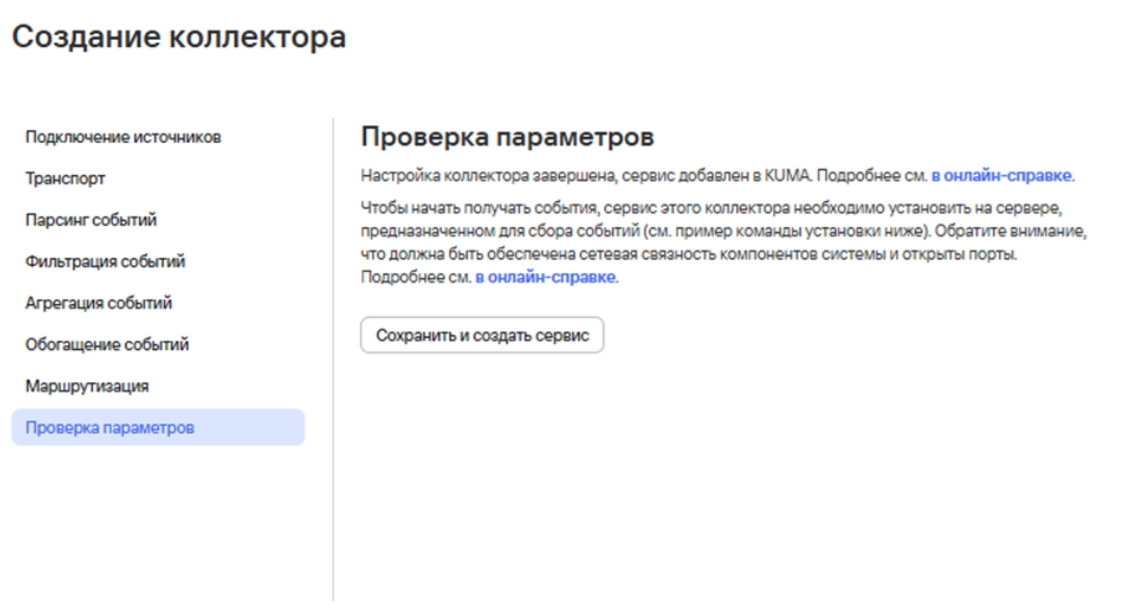
1. На вкладке «Парсинг событий» - «Настройки парсинга» выберите соответствующий нормализатор для обработки входящих событий.



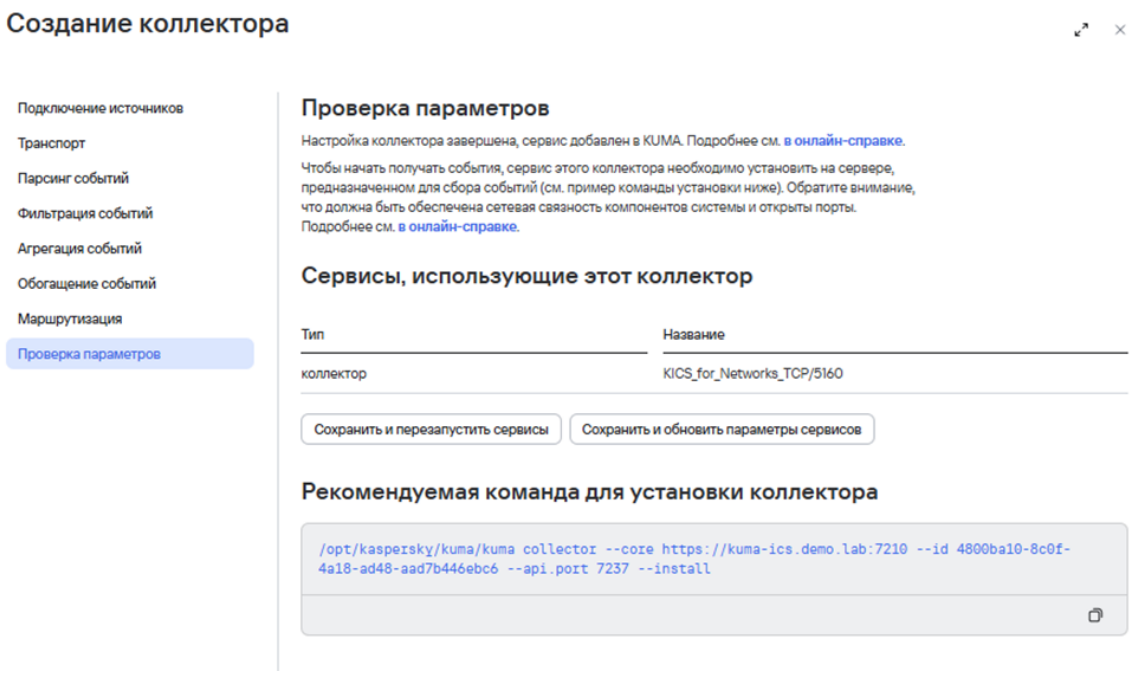
1. На вкладке «Маршрутизация» добавьте ранее созданные коррелятор и хранилище. Подробнее о развертывании этих компонентов см. в Справке по KUMA.



1. На вкладке «Проверка параметров» нажмите кнопку «Сохранить и создать сервис».

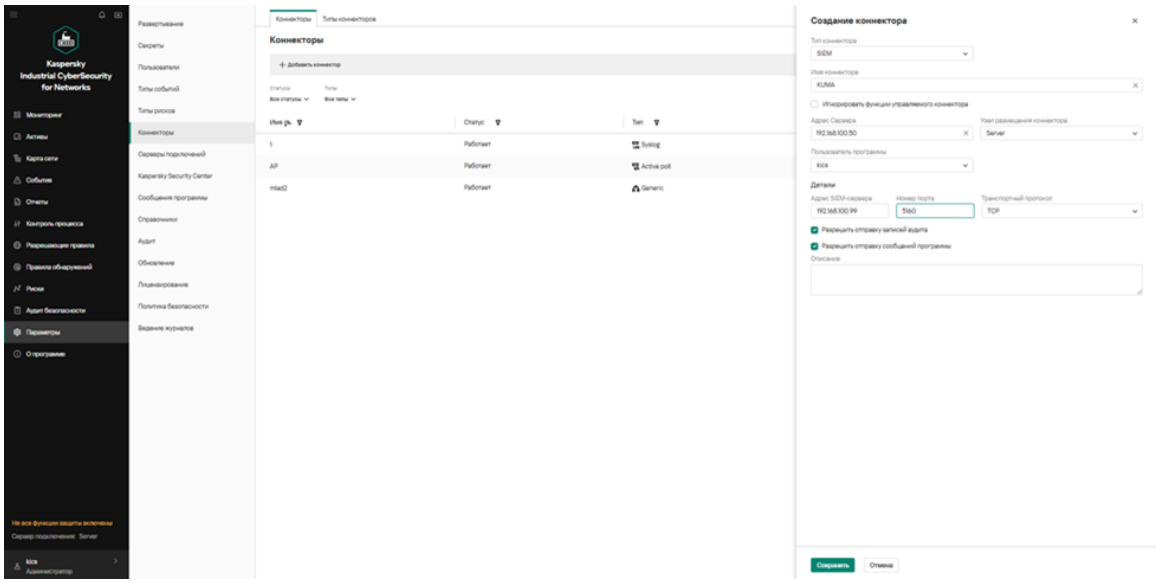


1. После создания коллектора скопируйте сгенерированную команду и выполните её в терминале сервера KUMA с правами администратора для развертывания сервиса.

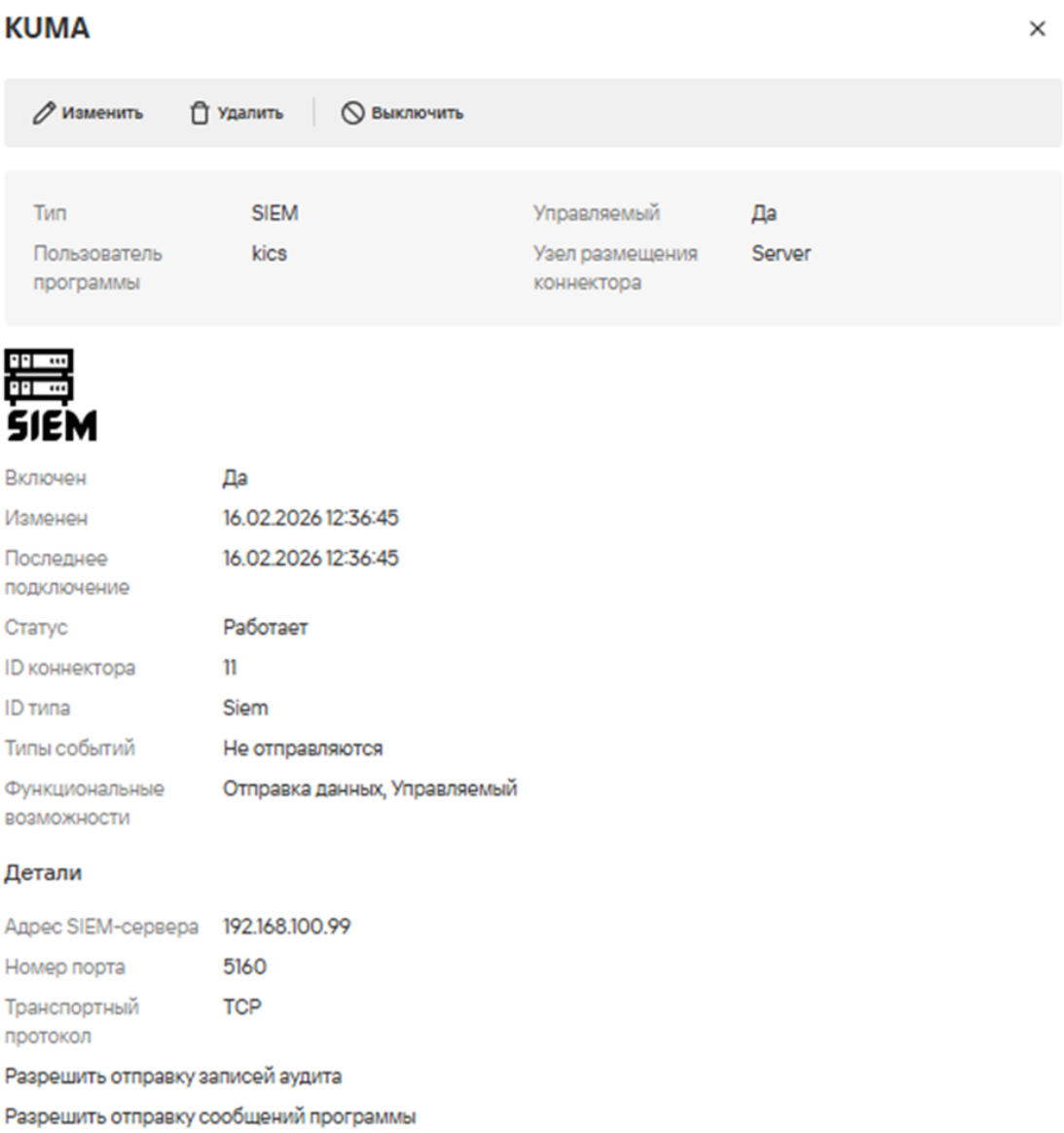


Настройка KICS for Networks

1. Перейдите в раздел «Параметры» - «Коннекторы» и нажмите кнопку «+ Добавить коннектор».
2. В карточке нового коннектора заполните поля:
   - Тип коннектора: SIEM.
   - Имя коннектора: Произвольное наименование.
   - Адрес сервера: IP-адрес узла, на котором развернут сервер KICS for Networks.
   - Узел размещения коннектора: Server.
   - Пользователь программы: Учетная запись администратора KICS for Networks.
   - Адрес SIEM-сервера: IP-адрес узла, на котором развернут сервер KUMA.
   - Номер порта: Порт, указанный при настройке коллектора KUMA (см. п. 4 раздела «Настройка KUMA»).
   - Транспортный протокол: Протокол, указанный при настройке коллектора KUMA (см. п. 4 раздела «Настройка KUMA»).

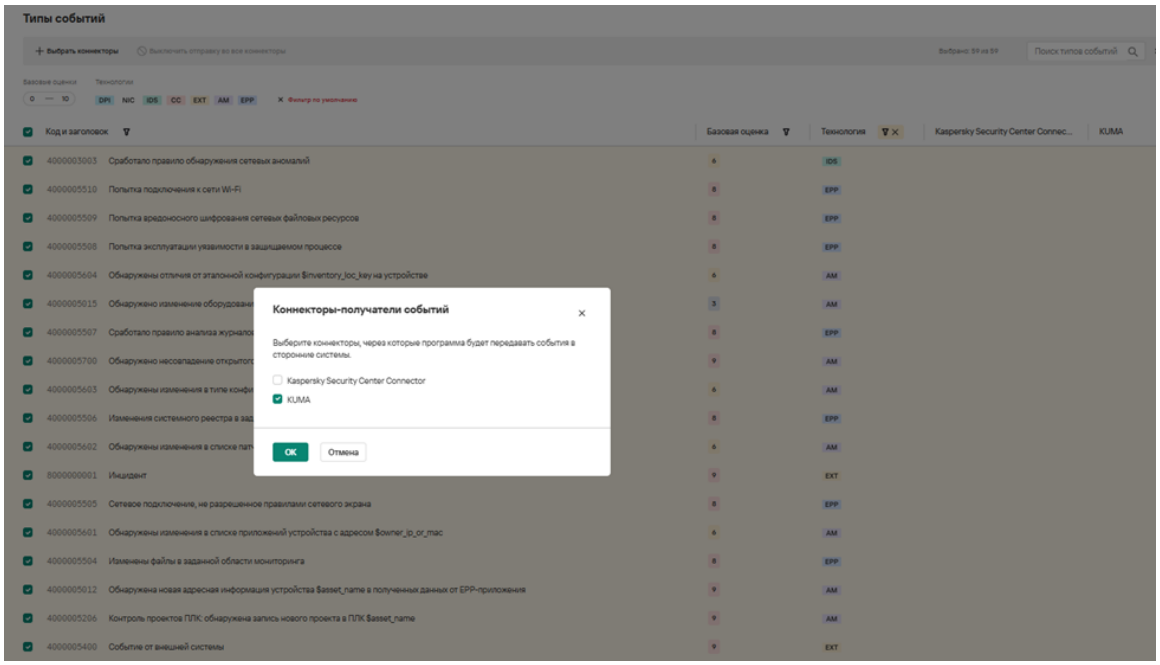


1. Нажмите кнопку «Сохранить».
2. Убедитесь, что в свойствах созданного коннектора параметры имеют следующие значения:
   - Включен: Да.
   - Статус: Работает.



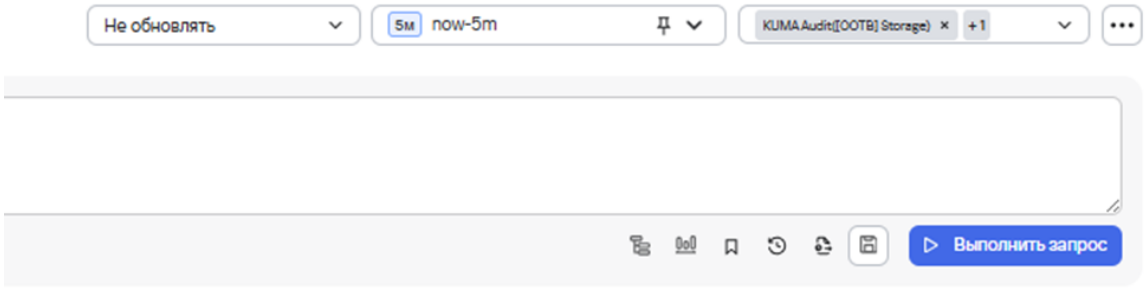
Если значения отличаются, проверьте выполненные настройки и перезапустите коннектор (выполните последовательность «Выключить» - «Включить»).

3. Перейдите в раздел «Параметры» - «Типы событий». Выберите типы событий, которые необходимо передавать в KUMA, и активируйте для них отправку через созданный коннектор.

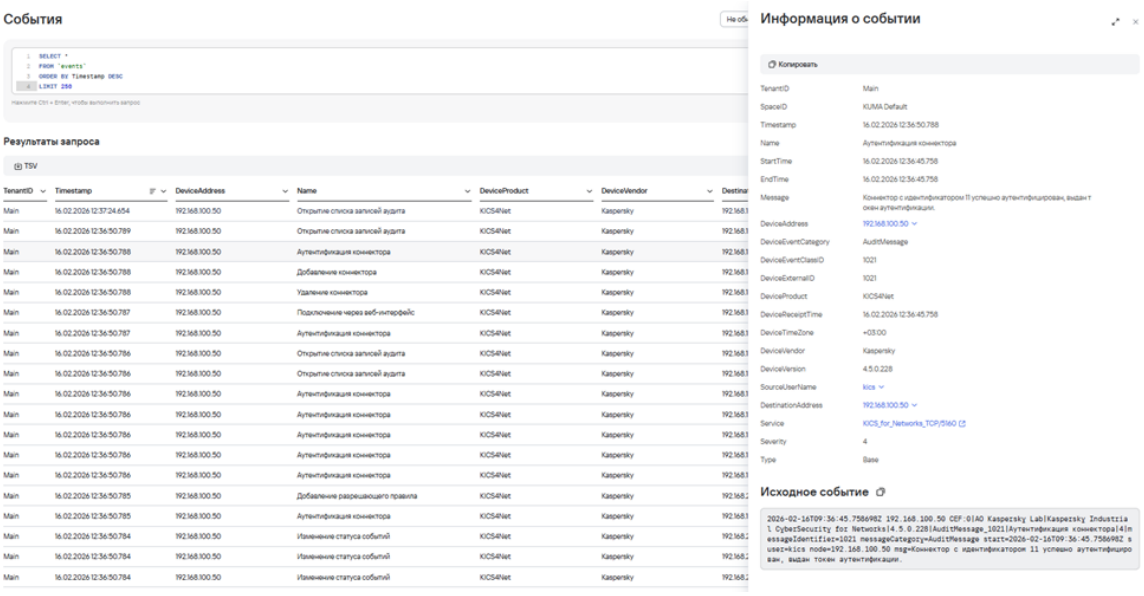


Проверка передачи событий

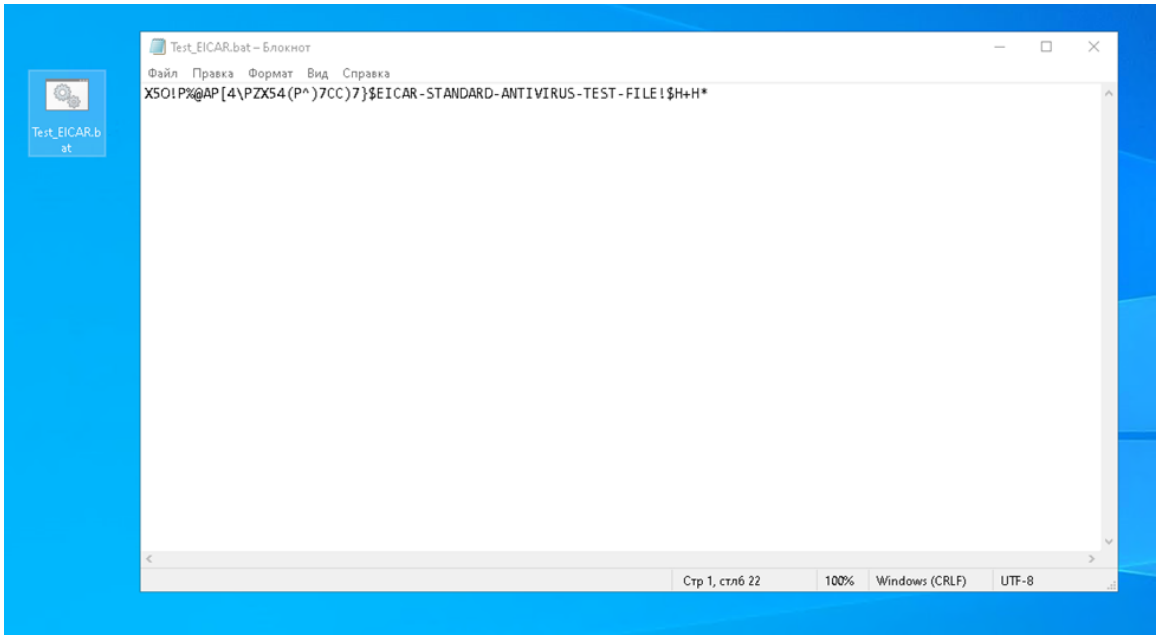
1. В интерфейсе KUMA перейдите в раздел «События» и выполните опрос источников.



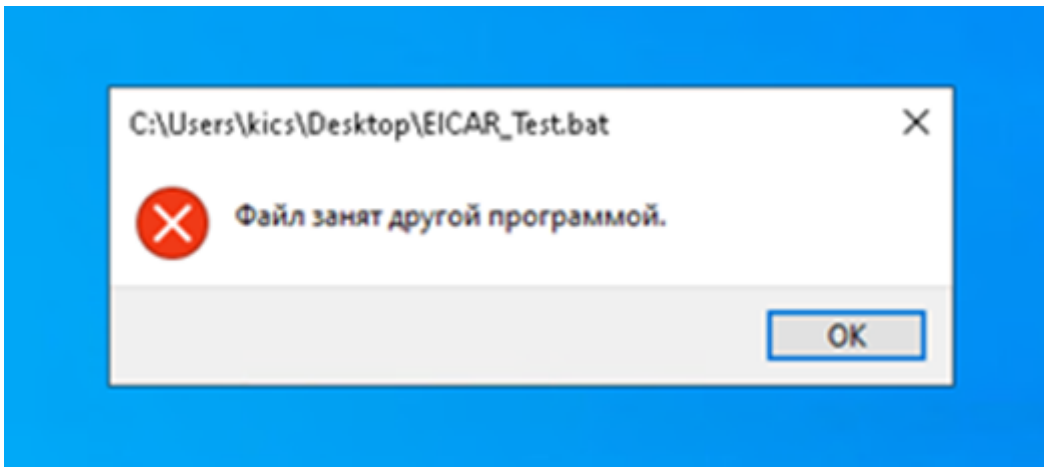
2. Убедитесь в наличии событие об успешной аутентификации коннектора, которое должно появиться в момент его создания в KICS for Networks.



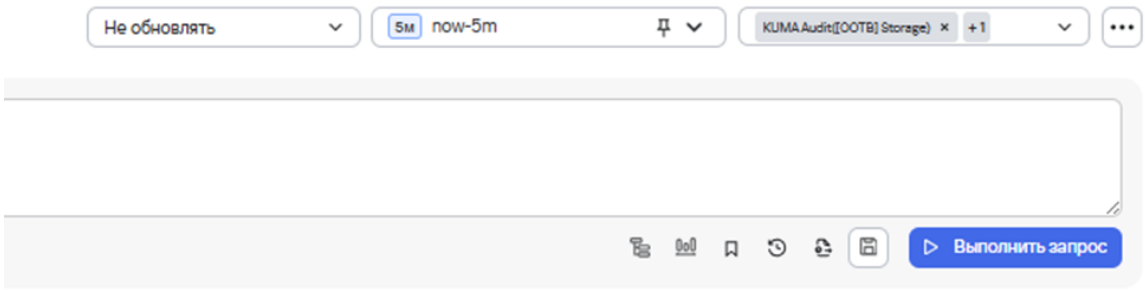
1. Воспроизведите событие безопасности, например, запустите тестовый вирус на узле, который интегрирован с KICS for Networks.



1. Убедитесь, что сработал модуль постоянной защиты KICS for Nodes (детектирование события на стороне источника).



1. В интерфейсе KUMA снова перейдите в раздел «События» и выполните опрос.



1. Убедитесь, что воспроизведенное событие безопасности успешно доставлено и отображается в KUMA.