

Архитектурные решения для обеспечения безопасности в условиях политики нулевого доверия

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация:

В статье рассматриваются архитектурные подходы к обеспечению информационной безопасности в условиях применения политики нулевого доверия (Zero Trust). Приводится анализ ключевых принципов модели Zero Trust и предлагаются архитектурные решения, направленные на усиление защиты информационных систем в современных условиях киберугроз.

Ключевые слова: нулевое доверие, архитектура безопасности, информационная безопасность, идентификация, аутентификация, контроль доступа, сегментация сети.

Architectural solutions for ensuring security in a zero trust environment

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article examines architectural approaches to ensuring information security in the context of applying the Zero Trust policy. The analysis of the key principles of the Zero Trust model is provided and architectural solutions aimed at enhancing the protection of information systems in the modern context of cyber threats are proposed.

Keywords: Zero Trust, security architecture, information security, identification, authentication, access control, network segmentation.

1. Введение

1.1 Актуальность

Рост цифровизации, распространение удалённой работы, развитие облачных сервисов и увеличение количества подключенных устройств привели к существенному усложнению периметра корпоративной сети. Традиционные модели безопасности, ориентированные на защиту периметра, становятся всё менее эффективными. Современные угрозы могут исходить как из внешней среды, так и изнутри организации, что требует переосмысления подходов к обеспечению безопасности. Политика нулевого доверия (Zero Trust) предлагает новый подход: никто и ничто не считается доверенным по

умолчанию. Все пользователи и устройства должны пройти строгую проверку перед получением доступа к ресурсам.

1.2 Обзор исследований

В статье, опубликованной на портале OSP.ru, рассматривается концепция нулевого доверия в контексте кибербезопасности [1]. Подчеркивается, что принцип нулевого доверия, предложенный в 2011 году компанией Forrester и Национальным институтом стандартов и технологий (NIST), основывается на предположении, что сеть всегда находится под угрозой.

В публикации А. И. Бражук, Е. В. Олизарович «К вопросу проектирования компьютерных систем на основе архитектуры нулевого доверия» подробно анализируется концепция архитектуры нулевого доверия (Zero Trust Architecture, ZTA) [2]. Отмечается, что ZTA представляет собой развивающуюся концепцию, для которой пока отсутствуют сертификация и практические стандарты, что создает путаницу среди вендоров и пользователей. Кроме того, перечислены ключевые компоненты модели нулевого доверия: Движок политик (Policy Engine), Администратор политик (Policy Administrator) и Точка принудительного исполнения (Policy Enforcement Point), а также вспомогательные системы, включая системы непрерывной диагностики (CDM), управления идентификацией и SIEM.

В статье С. В. Малинского «Концепция безопасности Zero Trust: принципы и практика внедрения» представлено определение концепции нулевого доверия и ее ключевые принципы [3]. Объясняется, что нулевое доверие предполагает отсутствие доверия к любым устройствам и пользователям по умолчанию, требуя постоянной проверки и авторизации. Статья также подчеркивает применимость подхода для организаций, использующих удаленную работу и гибридные облачные среды.

Согласно исследованию компании Microsoft (2021) [4], более 76% организаций во всём мире уже начали внедрение элементов Zero Trust. В докладе подчёркивается, что основные направления реализации — это многофакторная аутентификация, управление идентичностями и микросегментация сети.

1.3 Цель исследования

Цель исследования — продемонстрировать практическую реализацию политики нулевого доверия на примере простого веб-приложения, которое наглядно покажет, как пользователи получают доступ к данным при определенных правах.

Будет воссоздано простейшее REST API, которое реализует аутентификацию с помощью JSON Web Tokens (JWT), ограничение доступа на основе ролей пользователей, логирование всех действий и безопасное хранение конфиденциальных данных. Через серию тестов с использованием HTTP-запросов покажется, как приложение обеспечивает выполнение

принципов Zero Trust, включая строгую проверку каждого запроса и отказ в доступе при отсутствии необходимых прав.

2 Материалы и методы

Для реализации проекта используются следующие библиотеки Python:

1. Flask. Лёгкий веб-фреймворк для создания REST API.

Позволяет быстро настроить сервер, обрабатывать HTTP-запросы и реализовать эндпоинты для аутентификации и доступа к ресурсам.

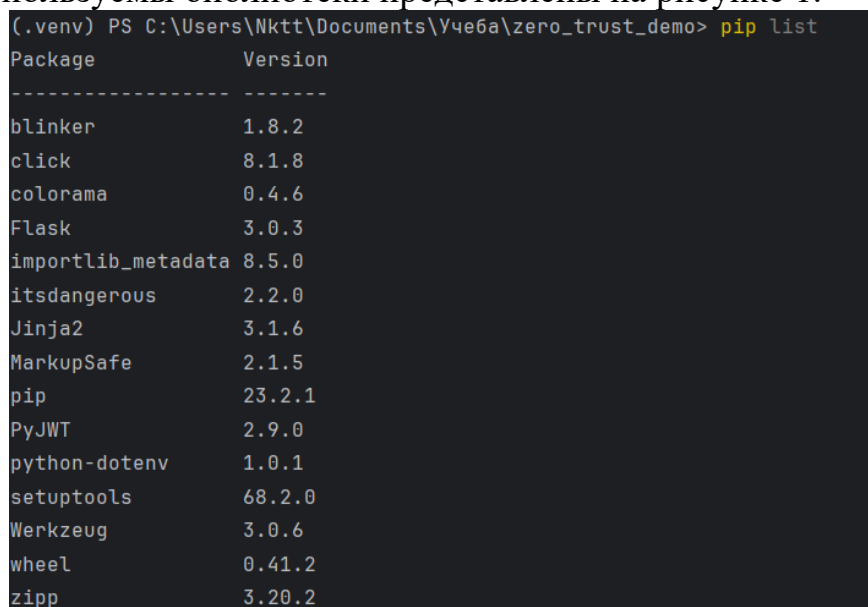
2. PyJWT. Библиотека для создания и проверки JSON Web Tokens (JWT).

Обеспечивает stateless-аутентификацию, позволяя кодировать данные пользователя (например, имя и роль) в токен, подписанный секретным ключом, и проверять его на каждом запросе.

3. Python-dotenv. Загрузка переменных окружения из файла .env.

Хранит секретный ключ для JWT в отдельном файле, обеспечивая безопасность конфиденциальных данных.

Все используемые библиотеки представлены на рисунке 1.



Package	Version
-----	-----
blinker	1.8.2
click	8.1.8
colorama	0.4.6
Flask	3.0.3
importlib_metadata	8.5.0
itsdangerous	2.2.0
Jinja2	3.1.6
MarkupSafe	2.1.5
pip	23.2.1
PyJWT	2.9.0
python-dotenv	1.0.1
setuptools	68.2.0
Werkzeug	3.0.6
wheel	0.41.2
zipp	3.20.2

Рисунок 1 – Список используемых библиотек

3 Результаты и обсуждения

3.1 Ожидаемые результаты

В этой практической части рассмотрим реализацию простого веб-приложения, демонстрирующего ключевые принципы политики нулевого доверия (Zero Trust). Zero Trust предполагает, что ни один пользователь или запрос не считается доверенным по умолчанию, каждый должен пройти строгую аутентификацию и авторизацию, а все действия должны фиксироваться для мониторинга. Мы создадим приложение на Python с использованием фреймворка Flask, реализующее аутентификацию через JWT-токены, ограничение доступа на основе ролей и логирование активности.

Целью является показать, как даже в минималистичной системе можно внедрить принципы Zero Trust: никому не доверять, постоянно проверять подлинность, минимизировать привилегии и отслеживать действия. Для этого

разработана API с тремя эндпоинтами: публичным, защищённым и доступным только администраторам.

3.2 Архитектура приложения

Приложение состоит из REST API, реализованного на Flask, с тремя основными эндпоинтами (рис.2):

1. /login (POST): выдаёт JWT-токен при успешной аутентификации.
2. /public (GET): публичный ресурс, доступный без аутентификации.
3. /protected (GET): защищённый ресурс, доступный только для аутентифицированных пользователей.
4. /admin (GET): ресурс, доступный только для пользователей с ролью admin.

Ключевые элементы Zero Trust в реализации:

1. Аутентификация: каждый запрос к защищённым эндпоинтам требует JWT-токена, который проверяется на валидность и срок действия.
2. Авторизация: доступ к /admin ограничен ролью admin, что реализует принцип наименьших привилегий (рис.4).
3. Мониторинг: все действия (успешные и неуспешные запросы) записываются в файл logs/access.log.
4. Безопасность конфигурации: секретный ключ для JWT хранится в файле .env.

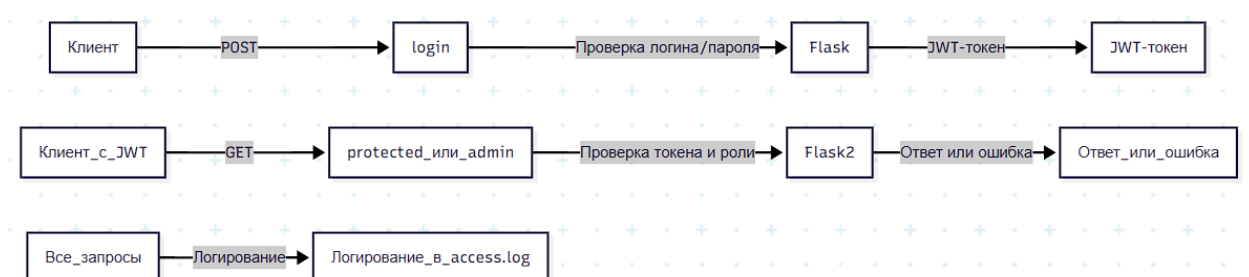


Рисунок 2 – Архитектура приложения

3.3 Код приложения

Логирование настроено для записи всех действий в файл logs/access.log, что позволяет отслеживать попытки доступа.

Секретный ключ для JWT загружается из .env с помощью python-dotenv, обеспечивая безопасное хранение конфиденциальных данных.

Пользователи хранятся в словаре USERS (для простоты; в реальной системе используется база данных) (рис.3).

```

1 import os
2 import jwt
3 import logging
4 from datetime import datetime, timedelta
5 from functools import wraps
6 from flask import Flask, request, jsonify
7 from dotenv import load_dotenv
8
9 # Настройка логирования
10 logging.basicConfig(filename='logs/access.log', level=logging.INFO,
11                     format='%(asctime)s - %(levelname)s - %(message)s')
12 # Инициализация Flask
13 app = Flask(__name__)
14 load_dotenv()
15 # Секретный ключ для JWT (хранится в .env)
16 SECRET_KEY = os.getenv('JWT_SECRET_KEY', 'your-very-secure-secret-key')
17 # Хардкод пользователей (в реальной системе – база данных)
18 USERS = {
19     'user1': {'password': 'pass123', 'role': 'user'},
20     'admin1': {'password': 'admin123', 'role': 'admin'}
21 }

```

Рисунок 3 – Настройка логирования и загрузка секретного ключа в app.py

```

def require_auth(role=None):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            token = request.headers.get('Authorization')
            if not token:
                logging.warning('Missing token')
                return jsonify({'error': 'Token is missing'}), 401

            try:
                # Удаляем префикс "Bearer " из заголовка
                token = token.replace('Bearer ', '')
                payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
                username = payload['username']
                user_role = payload['role']

                # Проверка существования пользователя
                if username not in USERS:
                    logging.warning(f'Invalid user: {username}')
                    return jsonify({'error': 'Invalid user'}), 403

                # Проверка роли
                if role and user_role != role:
                    logging.warning(f'Access denied for {username} with role {user_role}')
                    return jsonify({'error': 'Insufficient permissions'}), 403

                logging.info(f'Access granted for {username} with role {user_role}')
            except jwt.ExpiredSignatureError:
                logging.warning('Token expired')
                return jsonify({'error': 'Token expired'}), 401
            except jwt.InvalidTokenError:
                logging.warning('Invalid token')
                return jsonify({'error': 'Invalid token'}), 401
        return decorated_function
    return decorator

```

Рисунок 4 - Проверка доступа

Декоратор `require_auth` проверяет наличие JWT-токена в заголовке `Authorization`, его валидность и срок действия.

Если указана роль (например, admin) (рис.6), доступ предоставляется только пользователям с этой ролью, реализуя принцип наименьших привилегий.

Эндпоинт /login (рис.5) принимает JSON с логином и паролем, проверяет их и выдаёт JWT-токен с данными пользователя (имя, роль, срок действия).

Токен подписывается секретным ключом и имеет срок действия 15 минут, что соответствует принципу временного доступа.

```
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    if not username or not password:
        logging.warning('Missing username or password')
        return jsonify({'error': 'Missing username or password'}), 400

    user = USERS.get(username)
    if user and password == user['password']:
        # Создаем JWT с истечением через 15 минут
        token = jwt.encode({
            'username': username,
            'role': user['role'],
            'expiration': (datetime.datetime.now() + timedelta(minutes=15)).timestamp()
        }, SECRET_KEY, algorithm='HS256')
        logging.info(f'Login successful for {username}')
        return jsonify({'token': token})
    else:
        logging.warning(f'Invalid login attempt for {username}')
        return jsonify({'error': 'Invalid credentials'}), 401
```

Рисунок 5 – Эндпоинт /login

```

86 # Публичный эндпоинт (без авторизации)
87 @app.route('/public', methods=['GET'])
88 def public():
89     logging.info('Public endpoint accessed')
90     return jsonify({'message': 'This is public data'})
91
92 # Защищённый эндпоинт (доступ для всех аутентифицированных)
93 @app.route('/protected', methods=['GET'])
94 @require_auth()
95 def protected():
96     return jsonify({'message': 'This is protected data'})
97
98 # Эндпоинт только для админов
99 @app.route('/admin', methods=['GET'])
100 @require_auth(role='admin')
101 def admin():
102     return jsonify({'message': 'This is admin-only data'})
103

```

Рисунок 6 – Остальные эндпоинты

3.4 Тестирование приложения

Теперь покажем, как приложение реализует принципы Zero Trust на практике. Мы протестируем его с помощью команды curl, отправляя запросы к различным эндпоинтам. Получение JWT-токена показано на рис.7.

```

C:\Users\Wktt>curl -X POST http://localhost:5001/login -H "Content-Type: application/json" -d '{"username": "user1", "password": "pass123"}'
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InVzZXIiLCJleHBpcmF0aW9uIjoxNzUwNTI5Njg5LjIjXlN30.dXfq2z41xZG99mqbG1YjrBgXJvb8z67JMGVndLQDef8"
}

```

Рисунок 7 – получение JWT-токена

В результате запроса выдан токен, действующий 15 минут для пользователя user1, затем проверяем этот токен через запрос доступа к защищенному ресурсу (рис.8).

```

C:\Users\Wktt>curl http://localhost:5001/protected -H "Authorization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InVzZXIiLCJleHBpcmF0aW9uIjoxNzUwNTI5Njg5LjIjXlN30.dXfq2z41xZG99mqbG1YjrBgXJvb8z67JMGVndLQDef8"
{
  "message": "This is protected data"
}

```

Рисунок 8 – Доступ к защищенному ресурсу

Теперь попытаемся запросить доступ к админскому ресурсу с тем же токеном (рис.9).

```

C:\Users\Wktt>curl http://localhost:5001/admin -H "Authorization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InVzZXIiLCJleHBpcmF0aW9uIjoxNzUwNTI5Njg5LjIjXlN30.dXfq2z41xZG99mqbG1YjrBgXJvb8z67JMGVndLQDef8"
{
  "error": "Insufficient permissions"
}

```

Рисунок 9 – Попытка доступа к административному ресурсу

Пояснение: пользователь user1 с ролью user не имеет доступа к /admin, что демонстрирует принцип наименьших привилегий.

Также попытаемся получить доступ в систему без личного токена пользователя (рис.10).

```
C:\Users\Nktt>curl http://localhost:5001/protected
{
  "error": "Token is missing"
}
```

Рисунок 10 – Попытка доступа без токена

Пояснение к рисунку 10: отсутствие токена приводит к отказу в доступе, что соответствует принципу «никому не доверять».

Проверим логи, чтобы понять, пытался ли кто-то проникнуть в систему без прав доступа (рис.11).

```
* Running on http://127.0.0.1:5001
2025-06-22 13:48:22,743 - INFO - [33mPress CTRL+C to quit[0m
2025-06-22 13:48:22,745 - INFO - * Restarting with stat
2025-06-22 13:48:22,973 - WARNING - * Debugger is active!
2025-06-22 13:48:22,976 - INFO - * Debugger PIN: 117-738-146
2025-06-22 13:59:49,214 - INFO - Login successful for user1
2025-06-22 13:59:49,214 - INFO - 127.0.0.1 - - [22/Jun/2025 13:59:49] "POST /login HTTP/1.1" 200 -
2025-06-22 14:02:31,714 - INFO - Access granted for user1 with role user
2025-06-22 14:02:31,714 - INFO - 127.0.0.1 - - [22/Jun/2025 14:02:31] "GET /protected HTTP/1.1" 200 -
2025-06-22 14:16:45,514 - WARNING - Access denied for user1 with role user
2025-06-22 14:16:45,514 - INFO - 127.0.0.1 - - [22/Jun/2025 14:16:45] "[31m[1mGET /admin HTTP/1.1[0m" 403 -
2025-06-22 14:18:53,987 - WARNING - Missing token
2025-06-22 14:18:53,990 - INFO - 127.0.0.1 - - [22/Jun/2025 14:18:53] "[31m[1mGET /protected HTTP/1.1[0m" 401 -
```

Рисунок 11 – Логи из файла access.log

3.5 Ограничения и рекомендации для реальных систем

Эта реализация является упрощённой демонстрацией Zero Trust. Её ограничения:

1. Хардкод пользователей вместо базы данных.
2. Отсутствие HTTPS для шифрования соединения.
3. Нет многофакторной аутентификации (MFA).
4. Логирование ограничено локальным файлом.

Для реальных систем рекомендуется:

1. Использовать базу данных (например, PostgreSQL) для хранения пользователей.
2. Включить HTTPS с помощью SSL/TLS.
3. Добавить MFA (например, OTP через SMS или email).
4. Интегрировать с SIEM-системой для анализа логов.
5. Использовать прокси или WAF для защиты от атак.

4 Выводы

В процессе исследования было реализовано простое веб-приложение,

демонстрирующее принципы политики нулевого доверия: строгую аутентификацию с помощью JWT, ограничение доступа на основе ролей, мониторинг через логирование и безопасное хранение конфиденциальных данных. Тестирование показало, что система корректно обрабатывает запросы, предоставляя доступ только аутентифицированным пользователям с соответствующими правами и фиксируя все действия. Эта реализация может служить основой для более сложных систем, включающих дополнительные меры безопасности, такие как MFA и HTTPS. Данный проект также можно загрузить и ознакомиться с ним [6].

Библиографический список

1. Шор М., Зидэлли Ш., Кешария А. Принцип нулевого доверия: что, как, почему, когда? //Открытые системы. СУБД. №. 1. С. 17-22.
2. Бражук А. И., Олизарович Е. В. К вопросу проектирования компьютерных систем на основе архитектуры нулевого доверия //Информатика. 2024. Т. 21. №. 4. С. 85-98.
3. Малинский С. В. Концепция безопасности Zero Trust: принципы и практика внедрения //Интеллектуальные транспортные системы. – 2022. – С. 430-437.
4. Microsoft Corporation. The State of Zero Trust Report 2021. Microsoft, 2021. URL: <https://www.microsoft.com/security/blog/2021/07/29/the-state-of-zero-trust> (дата обращения: 08.06.2025)
5. National Institute of Standards and Technology. Special Publication 800-207: Zero Trust Architecture / Rose S., Borchert O., Mitchell S., Connelly S. – Gaithersburg, MD: NIST, 2020. – 59 с. – DOI: 10.6028/NIST.SP.800-207.
6. Проект на Google disk. URL: https://drive.google.com/drive/folders/15q2bgwBm-cfHO6qwMx8fU-18CJ1Sq_EG?usp=sharing