Механизмы межсервисной аутентификации в приложениях с микросервисной архитектурой

К.И. Зимина, О.Р. Лапонина

Аннотация **– В данной статье обсуждается тема межсервисной аутентификации, которая является одним из важных аспектов безопасности в современных приложениях с микросервисной архитектурой.**

В статье представлены основные механизмы работы service-to-service аутентификации, включая использование токенов и сертификатов. Приводится пример архитектуры сервис-сервисной аутентификации.

Статья является полезным ресурсом для разработчиков, которые работают в области сервис-ориентированных архитектур и интересуются вопросами безопасности. Она предоставляет обширный обзор механизмов аутентификации и основные моменты, на которые следует обратить внимание при проектировании микросервисной архитектуры.

В статье предлагается использовать протокол mutual TLS (mTLS) как наиболее популярный способ обеспечения безопасности межсервисной связи при развертывании микросервисов.

В данном подходе ответственность за межсервисную аутентификацию лежит на mTLS прокси, развернутом около каждого микросервиса системы. mTLS прокси работают как посредники между микросервисами, принимая запросы на установление защищенного канала связи. Подход с использованием прокси позволяет упростить процесс проверки подлинности между двумя микросервисами, которые могут работать на разных платформах, с использованием разных протоколов и форматов данных.

Благодаря использованию mTLS прокси решение легко масштабировать, так как при появлении новых микросервисов в системе достаточно лишь развернуть новый экземпляр mTLS прокси. Также прокси никак не зависит от языка или системы, с помощью которых был реализован связанный с ним микросервис, что делает решение универсальным.

Ключевые слова **– микросервисная архитектура, аутентификация, доверенные сети, mTLS, JWT, PKI.**

I. Введение

В условиях быстрого развития технологий и увеличения объемов обрабатываемых данных, микросервисная архитектура становится все более популярной среди разработчиков. Микросервисный подход к разработке программного обеспечения подразумевает декомпозицию приложения на множество небольших сервисов, каждый из которых отвечает за определенную функциональность. Это позволяет ускорить разработку, улучшить масштабируемость и гибкость приложения [1].

Взаимодействие микросервисов — это ключевой аспект микросервисной архитектуры, который обеспечивает работу приложения в целом. Каждый микросервис представляет собой небольшое приложение, которое выполняет определенную функцию и коммуницирует с другими сервисами для выполнения конкретной задачи. Крайне важно обеспечить надежное и защищенное взаимодействие в микросервисной архитектуре, для того чтобы безопасно ее использовать [2].

Наиболее ответственным этапом взаимодействия является установка соединения, во время которого происходит процедура аутентификации сторон [3]. Service-to-service аутентификация (аутентификация между сервисами) — это процесс проверки подлинности между двумя сервисами в рамках микросервисной архитектуры приложений. Этот процесс обеспечивает безопасную передачу данных между сервисами и защищает их от несанкционированного доступа.

Проектирование механизмов сервис-сервисной аутентификации может оказаться весьма трудоемким процессом, так как архитекторам проекта требуется учесть множество различных факторов [4], [5], [6]:

- взаимодействующие микросервисы могут быть реализованы на разных платформах, с использованием различных языков программирования;
- механизм аутентификации должен быть безопасным и устойчивым к атакам типа перехват трафика, подмена токенов и т.п;
- может потребоваться настройка сложной инфраструктуры, такой как система управления ключами.

Учитывая важность вопроса межсервисной аутентификации, разработчики должны тщательно планировать и организовывать взаимодействие микросервисов, чтобы обеспечить безопасность приложения.

Целью данного исследования было составление наиболее общего представления о механизмах

Статья получена 20 апреля 2023.
К. И. Зимина, МГУ им. М.В. Ломоносова (e-mail: xeniazimina2000@gmail.com)
О.Р. Лапонина, МГУ им. М.В. Ломоносова (e-mail: laponina@oit.cmc.msu.ru).

аутентификации микросервисов. В статье рассматриваются различные подходы к проектированию механизмов аутентификации. В частности, приводится пример архитектуры микросервисного приложения, который позволит облегчить настройку межсервисной аутентификации.

II. Основы микросервисной архитектуры

Система приложений на основе микросервисов состоит из нескольких компонентов (микросервисов), которые взаимодействуют друг с другом посредством синхронных удаленных вызовов процедур или асинхронной системы обмена сообщениями [7]. Каждый микросервис обычно реализует один (редко два и более) отдельных бизнес-процесса или определенную функциональность (например, хранение данных о клиенте, хранение и отображение каталога продукции, обработка заказов клиентов и т.д.) [8].

Некоторые сервисы могут предоставлять (REST)full API, которые используются другими микросервисами или клиентскими приложениями. Другие микросервисы могут реализовывать веб-интерфейс пользователя (UI).

Микросервисы могут развертываться различными способами. Это может быть процесс на сервере приложений, на виртуальной машине или в контейнере.

В основе разработки микросервисных приложений лежат следующие принципы [9]:

- каждый микросервис должен управляться, реплицироваться, масштабироваться, обновляться и развертываться независимо от других микросервисов;
- каждый микросервис должен выполнять одну функцию и работать в ограниченном контексте (т.е. иметь ограниченную ответственность и зависимость от других сервисов);
- все микросервисы должны быть отказоустойчивы и должны иметь возможность быстро восстанавливаться;
- для управления состоянием рекомендуется использовать существующие доверенные службы (например, базы данных, кэши и каталоги).

III. Типы межсервисных взаимодействий в микросервисной архитектуре

Приложения на основе микросервисов не ограничиваются какой-либо конкретной технологией и состоят из небольших независимых сущностей (конечных точек), которые взаимодействуют друг с другом с помощью облегченных механизмов. Эти конечные точки реализованы с использованием четко документированных API. Существует несколько типов API конечных точек, таких, как SOAP (Simple Object Access Control) или REST (Hypertext Transfer Protocol (HTTP)) [9]. Доступ к услугам предоставляется различными платформами или типами клиентов, например, веб-браузерами или мобильными устройствами, с использованием компонента, называемого «клиентом».

Для определения службы используется язык описания интерфейса (IDL) (например, Swagger/OpenAPI). Первый шаг в разработке сервиса включает в себя определение интерфейса, которое рассматривается с разработчиками клиентов и согласовывается до начала реализации сервиса. Таким образом, API служит контрактом между клиентами и сервисами.

Выбор механизма IPC (механизм межсервисного взаимодействия) определяет тип API.

Таблица 1. Соответствие между механизмом IPC и API

IPC механизм	API
Асинхронный, основанный на сообщениях (например, Advanced Message Queuing Protocol (AMQP) или Simple (или Streaming) Text Oriented Messaging (STOMP))	Состоит из каналов сообщений и типов сообщений
Синхронный запрос/ответ (например, основанный на HTTP REST или Thrift)	Состоит из URL-адресов и формата запросов и ответов

При реализации IPC могут использоваться различные типы форматов сообщений: текстовые, например, JavaScript Object Notation (JSON) или Extensible Markup Language (XML), либо двоичные форматы, например, буферы Apache Avro или Protocol.

A. Запрос-ответ

Можно определить два различных шаблона запросов, которые включают в себя запросы без поддержки состояния и команды для бизнес-функции с поддержкой состояния.

В первом шаблоне микросервис делает конкретный запрос на информацию, либо предпринимает какие-либо действия и ожидает ответ.

Во втором шаблоне один микросервис обращается к другому, чтобы тот предпринял некоторые действия, связанные с бизнес-функцией, меняющей состояние. В типе запрос-ответ существует сильная зависимость времени выполнения между двумя задействованными микросервисами, которая проявляется следующими двумя способами:

- один микросервис может выполнять свою функцию только тогда, когда доступен другой микросервис;
- микросервис, осуществляющий запрос, должен убедиться, что запрос был успешно доставлен целевому микросервису

Из-за характера связи в протоколе запрос-ответ используется протокол синхронной связи, такой как HTTP. Если микросервис реализован с помощью REST API, сообщения между микросервисами называются HTTP REST API. REST API часто определяются с использованием стандартного языка RAML (RESTful API Modeling Language), который был разработан для определения и объявления интерфейса микросервиса. HTTP — это блокирующий тип связи, который инициирует запрос и может продолжить свое выполнение только тогда, когда он получит ответ.

Простейший пример REST API запроса может

выглядеть следующим образом.

```
GET /api/users/1 HTTP/1.1
Host: example.com
Accept: application/json
```

Листинг 1. Пример REST API запроса

Этот запрос запрашивает данные пользователя с идентификатором 1 из REST API, размещенного на домене `example.com`.

Ответ на этот запрос может быть представлен в виде JSON-объекта.

```
{
     "id": 1,
     "name": "John Doe",
     "email": "john.doe@example.com"
}
```

Листинг 2. Ответ на запрос в виде JSON-объекта

Еще один пример взаимодействия типа запрос-ответ — SOAP-запрос. Он может выглядеть следующим образом.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns:ns="http://example.com/">
     <soap:Header/>
     <soap:Body>
         <ns:GetUserRequest>
             <ns:UserId>1</ns:UserId>
         </ns:GetUserRequest>
     </soap:Body>
</soap:Envelope>
```

Листинг 3. Пример SOAP-запроса

Этот SOAP-запрос запрашивает данные пользователя с идентификатором 1. Он использует пространство имен `ns` для определения элементов запроса `GetUserRequest` и `UserId`. Элемент `UserId` содержит значение 1.

Ответ на этот запрос может быть представлен в виде XML-документа.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns:ns="http://example.com/">
     <soap:Header/>
     <soap:Body>
         <ns:GetUserResponse>
             <ns:UserId>1</ns:UserId>
             <ns:UserName>John
         Doe</ns:UserName>
             <ns:UserEmail>
         john.doe@example.com</ns:UserEmail>
           </ns:GetUserResponse>
      </soap:Body>
</soap:Envelope>
```

Листинг 4. Пример ответа в виде XML-документа

SOAP-ответ содержит данные запрошенного пользователя, включая идентификатор, имя и адрес электронной почты.

B. Публикация-подписка

Этот тип используется, когда микросервисам необходимо взаимодействовать для реализации сложного бизнес-процесса или транзакции. «Публикацию-подписку» иначе называют подходом, основанным на событиях бизнес-домена, или подходом к подписке на события домена.

В данном шаблоне микросервис регистрируется или подписывается на события бизнес-домена (например, заинтересованные в конкретной информации или способные обрабатывать определенные запросы), которые публикуются брокеру сообщений через интерфейс шины событий. Эти микросервисы построены с использованием API, управляемых событиями, и используют асинхронные протоколы обмена сообщениями, такие как Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP) и Kafka Messaging, которые обеспечивают поддержку уведомлений и подписок. В асинхронных протоколах отправитель сообщения обычно не ждет ответа, а просто отправляет сообщение агенту сообщений (например, в очередь RabbitMQ [10]). Одним из сценариев использования этого подхода является распространение обновлений данных на несколько микросервисов на основе определенных событий.

Пример использования RabbitMQ [11] для асинхронной отправки сообщений на языке Python может выглядеть следующим образом.

```
import pika

# Подключаемся к RabbitMQ
connection                                    =
pika.BlockingConnection(pika.ConnectionParameters(
'localhost'))
channel = connection.channel()

# Создаем очередь для отправки сообщений
channel.queue_declare(queue='hello')
# Отправляем сообщение в очередь
channel.basic_publish(exchange='',
routing_key='hello', body='Hello, World!')

print(" [x] Sent 'Hello, World!'")

# Закрываем соединение
connection.close()
```

Листинг 5. Пример использования RabbitMQ для асинхронной отправки сообщений на языке Python

Этот код подключается к RabbitMQ-брокеру на локальном хосте, создает очередь с именем "hello" и регистрирует обработчик сообщений. Когда сообщение поступает в очередь, обработчик вызывается и выводит содержимое сообщения в консоль.

Пример использования Apache Kafka для асинхронной отправки и получения сообщений на языке Python может выглядеть следующим образом.

```
from kafka import KafkaProducer, KafkaConsumer

# Определяем параметры подключения к Kafka-брокеру
bootstrap_servers = ['localhost:9092']

# Создаем Kafka-продюсера и отправляем сообщение в тему
producer = KafkaProducer(bootstrap_servers=bootstrap_servers)
producer.send('test-topic', b'Hello, World!')

# Создаем Kafka-консьюмер и читаем сообщения из темы
consumer = KafkaConsumer('test-topic', bootstrap_servers=bootstrap_servers, auto_offset_reset='earliest', group_id=None)
for message in consumer:
print(message.value.decode())
```

Листинг 6. Пример использования Apache Kafka для асинхронной отправки и получения сообщений на языке Python

Этот код создает Kafka-продюсера, который отправляет сообщение "Hello, World!" в тему "test-topic". Затем он создает Kafka-консьюмер, который читает сообщения из этой темы и выводит содержимое сообщений в консоль.

IV. Механизмы сервис-сервисной аутентификации

Вне зависимости от того, какой был выбран тип межсервисного взаимодействия, для обеспечения безопасности этого взаимодействия могут быть использованы следующие подходы **Ошибка! Источник ссылки не найден.** :

- доверенные сети;
- mTLS;
- JWT.

A. Доверенные сети

Подход с использованием доверенных сетей возник раньше всех других подходов. Предполагается, что безопасность во время межсервисного взаимодействия никак не обеспечивается. Модель опирается на безопасность системы на сетевом уровне. Архитектура подхода с использованием доверенных сетей представлена на рисунке.

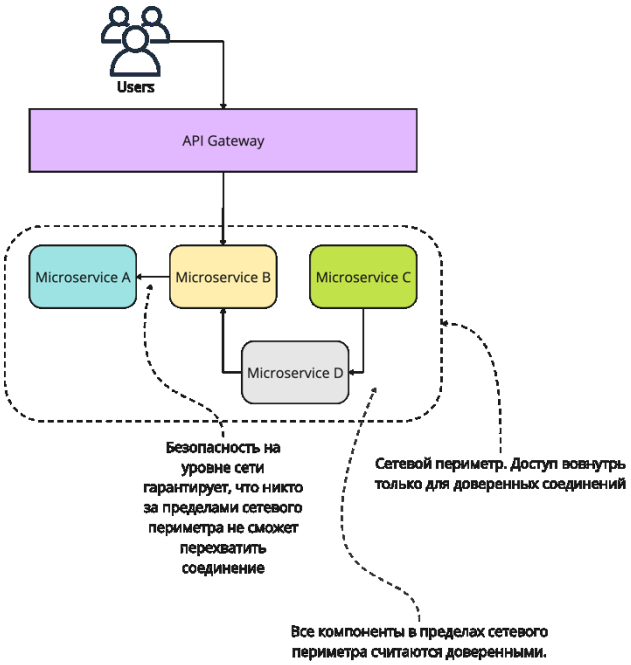

Рисунок 1. Архитектура подхода с использованием доверенных сетей

Безопасность на сетевом уровне в таком случае гарантирует, что злоумышленник не сможет перехватывать трафик между микросервисами. При этом каждый микросервис является надежной системой. Это означает, что всему, что микросервис утверждает о себе и о конечном пользователе, другие микросервисы доверяют и никак не проверяют.

Подобный подход не популярен, но тем не менее, используется при высоком уровне доверия к каждому компоненту приложения.

Сетевой подход с нулевым доверием является противоположным данному подходу. Он предполагает, что сетевая среда ненадежна и враждебна. Любая информация тщательным образом проверяется перед тем, как ей начнут доверять. Каждый запрос должен быть аутентифицирован и авторизован на каждом узле, прежде чем он будет принят для дальнейшей обработки [13].

B. Mutual TLS (mTLS)

Mutual TLS (mTLS) — еще один популярный способ обеспечения безопасности межсервисной связи при развертывании микросервисов **Ошибка! Источник ссылки не найден.**. Схема установки соединения представлена на рисунке.

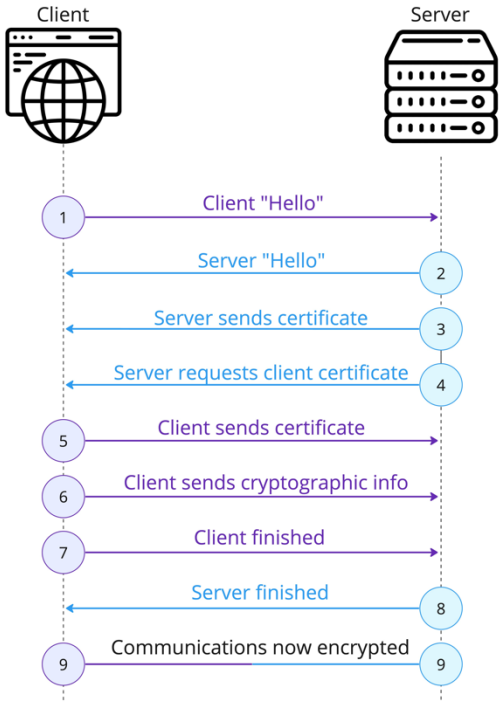

Рисунок 2. Схема установки mTLS соединения

Данный метод является наиболее распространенной формой межсервисной аутентификации, используемой сегодня.

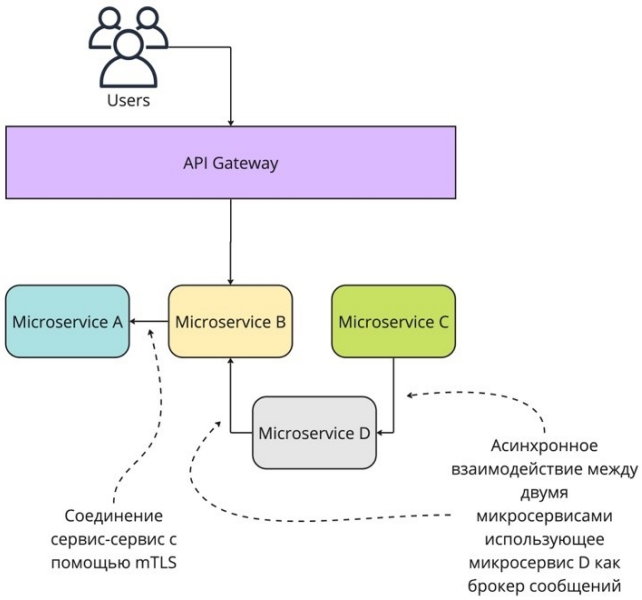

Рисунок 3. Архитектура подхода с использованием mTLS

Каждый микросервис при развертывании должен иметь открытый и закрытый ключи, которые будут использоваться для аутентификации при взаимодействии с другими микросервисами через mTLS. TLS обеспечивает конфиденциальность и целостность передаваемых данных, а также идентифицирует сервис.

В mTLS каждая сторона имеет свой сертификат, который используется для проверки подлинности. Микросервис-клиент отправляет свой сертификат микросервису-серверу, а микросервис-сервер отправляет свой сертификат микросервису-клиенту. Каждая сторона проверяет сертификат другой стороны, чтобы убедиться в том, что она является легитимным участником сессии. Если процесс аутентификации завершается успешно, устанавливается защищенный канал связи между микросервисом-клиентом и микросервисом-сервером.

Отличия TLS и mTLS представлены в таблице.

Таблица 2. Отличия TLS и mTLS

TLS	mTLS
Transport Layer Security (Безопасность на транспортном уровне)	Mutual transport layer security (Взаимная безопасность на транспортном уровне)
Только сервер аутентифицирует себя	И клиент, и сервер аутентифицируют себя

C. Json Web Tokens (JWT)

Веб-токен JSON — это третий подход к обеспечению безопасности межсервисной связи при развертывании микросервисов. Состав JWT представлен на рисунке.

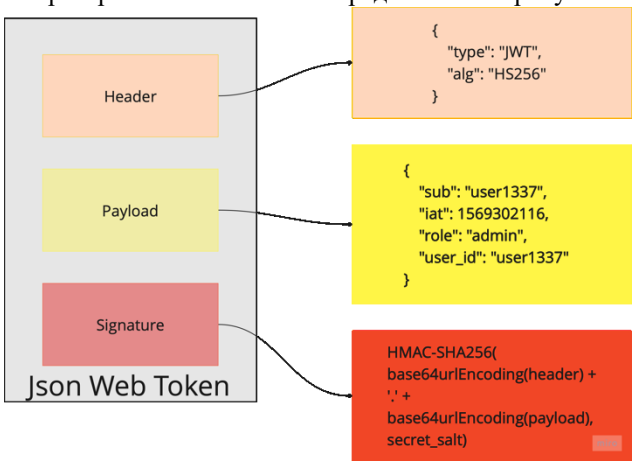

Рисунок 4. Состав JWT

В отличие от mTLS, JWT работает на прикладном уровне, а не на транспортном уровне эталонной модели ISO/OSI **Ошибка! Источник ссылки не найден.**. JWT — это контейнер, который может передавать набор утверждений из одного места в другое. Простейший пример архитектуры, использующей подход JWT представлен на рисункеРисунок 5.

В качестве утверждений могут выступать, например, атрибуты конечного пользователя (адрес электронной почты, номер телефона) или права конечного пользователя. JWT включает в себя эти поля и подписывается эмитентом JWT. Эмитентом может быть внешняя служба токенов безопасности (STS) или сам вызывающий микросервис.

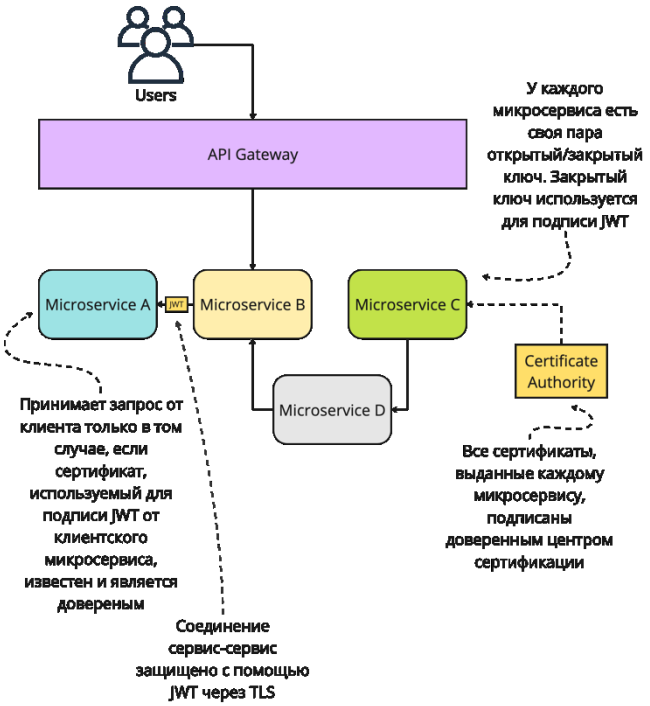

Рисунок 5. Архитектура подхода с использованием JWT

Существуют самоподписанные токены JWT. Как и в случае с mTLS, если используется самоподписанная аутентификация на основе JWT, каждый микросервис должен иметь свою собственную пару ключей, а соответствующий закрытый ключ используется для подписи JWT. В большинстве случаев аутентификация на основе JWT работает через TLS; JWT при этом обеспечивает аутентификацию, а TLS обеспечивает конфиденциальность и целостность передаваемых данных.

V. Архитектура межсервисной аутентификации в микросервисах

Для микросервисов, которые имеют доступ к защищенным ресурсам или вынуждены обмениваться конфиденциальной информацией, будь то персональные данные пользователей, сведения о товарах, заказах и платежах, наиболее предпочтительным методом сервис-сервисной аутентификации является mTLS. С точки зрения безопасности, в отличие от подхода с JWT, mTLS обеспечивает аутентификацию обеих сторон.

Однако, подход, использующий mTLS, гораздо сложнее реализовать, чем JWT, потому что он требует более сложной инфраструктуры сертификации и настройки сетевых устройств. Для работы mTLS необходимо настроить и установить сертификаты на клиентской и серверной стороне, а также настроить сетевое оборудование, чтобы оно поддерживало протоколы TLS и mTLS.

Для упрощения развертывания mTLS предлагается использовать mTLS прокси, как показано на рисункеРисунок 6.

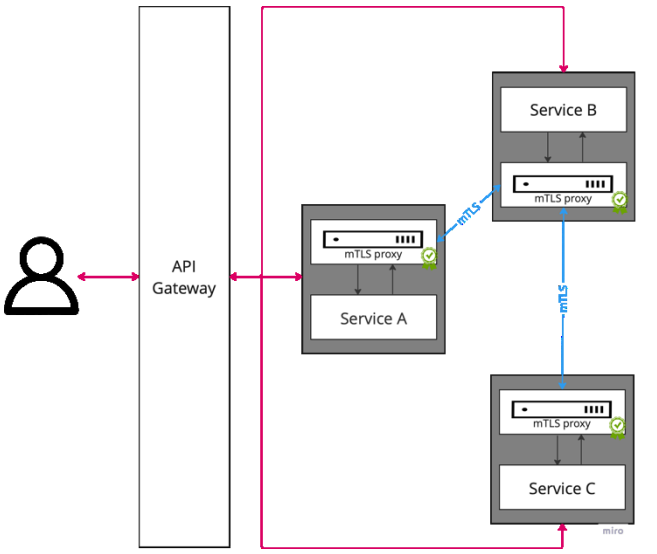

Рисунок 6. Архитектура межсервисной аутентификации с использованием mTLS прокси

В данном подходе ответственность за межсервисную аутентификацию лежит на mTLS прокси, развернутом около каждого микросервиса системы. mTLS прокси работают как посредники между микросервисами, принимая запросы на установление защищенного канала связи. Подход с использованием прокси позволяет упростить процесс проверки подлинности между двумя микросервисами, которые могут работать на разных платформах, с использованием разных протоколов и форматов данных.

Для установления безопасного канала связи mTLS прокси использует сертификаты и проверяют подлинность сторон. Каждый микросервис должен иметь свой собственные сертификат, который подтверждает его подлинность. Если сертификат не прошел проверку, mTLS прокси не позволит установить защищенное соединение.

Наиболее важной составляющей при развертывании подобного рода архитектуры является инфраструктура открытых ключей.

VI. Инфраструктура открытых ключей

Инфраструктура открытых ключей (Public Key Infrastructure, PKI) построена на наборе компонентов и процедур для управления парами открытых и закрытых ключей.

Типичная архитектура PKI состоит из следующих компонентов [16]:

- **Сертификат** — цифровой документ, подписанный центром сертификации и используемый для подтверждения владельца открытого ключа в PKI. Сертификат имеет ряд атрибутов, таких как серийный номер, срок действия, криптографические алгоритмы и параметры шифрования. Сертификат также содержит имя субъекта, которое является информацией, идентифицирующей владельца. Это может быть, например, DNS-имя или IP-адрес.
- **Пара открытый/закрытый ключ**. Закрытый ключ и связанный с ним открытый ключ математически связаны друг с другом. Открытый ключ свободно распространяется в виде сертификата открытого

ключа. Закрытый ключ подтверждает право собственности на личность и должен храниться в секрете у микросервиса или у соответствующего ему прокси.

- **Центр сертификации (Certificate Authority, CA)** выдает сертификат объекта и действует как доверенный компонент в PKI. Любому сертификату, выданному центром сертификации, доверяют все объекты, у которых есть открытый ключ данного CA.
- **Список отозванных сертификатов (Certificate Revocation List, CRL)** — список TLS-сертификатов, отозванных центром сертификации (CA). После поступления запроса на отзыв сертификата CA заносит уникальный серийный номер сертификата в список CRL, который:
    1. защищен цифровой подписью центра сертификации – не может быть изменен никем, кроме центра сертификации;
    2. обновляется минимум один раз в течение суток – содержит актуальный список отозванных центром сертификации сертификатов.

В следующем пункте приводится абстрактный пример процесса проверки сертификата для описанной выше архитектуры межсервисной аутентификации.

A. Процесс проверки сертификата

Процесс проверки сертификата представлен на рисунке. Certification Authority Server – это компонент инфраструктуры открытых ключей, который выполняет роль центра сертификации. Закрытые ключи микросервисов A и B хранятся у соответствующих им mTLS прокси.

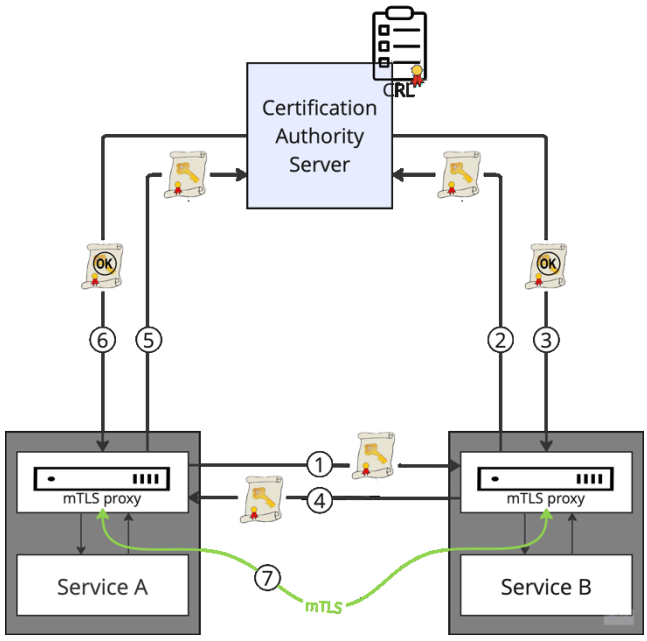

Рисунок 7. Процесс проверки сертификата для архитектуры межсервисной аутентификации с использованием mTLS прокси

Алгоритм процесса проверки сертификата на этапе установки соединения:

1. mTLS прокси микросервиса A отправляет свой сертификат mTLS прокси микросервиса B.
2. Прокси микросервиса B обращается к серверу центра сертификации с сертификатом микросервиса A для проверки его подлинности. Сервер центра сертификации проверяет, не присутствует ли сертификат в списке отозванных сертификатов (CRL), а также проверяет подлинность сертификата микросервиса A.
3. После проведения необходимых проверок сервер центра сертификации присылает прокси микросервиса B информацию об успешном/неуспешном результате проверок.
4. В случае успеха прокси микросервиса B в свою очередь отсылает свой сертификат прокси микросервиса A.
5. Аналогично, прокси микросервиса A обращается к серверу центра сертификации с сертификатом микросервиса B.
6. После проведения необходимых проверок сервер центра сертификации присылает прокси микросервиса A информацию об успешном/неуспешном результате проверок.
7. В случае успешного прохождения каждого этапа устанавливается безопасное соединение между mTLS прокси микросервиса A и микросервиса B, а значит, установилось соединение между микросервисами A и B.

Таким образом, получилась простейшая архитектура сервис-сервисной аутентификации в микросервисах. Благодаря использованию mTLS прокси решение легко масштабировать, так как при появлении новых микросервисов в системе достаточно лишь развернуть новый экземпляр mTLS прокси. Также прокси никак не зависит от языка или системы, с помощью которых был реализован связанный с ним микросервис, что делает решение универсальным. При этом, для взаимодействия микросервисов используется TLS, который обеспечивает конфиденциальность и целостность передаваемых данных.

Все вышеизложенное делает подобную архитектуру в достаточной мере безопасной для использования при передаче конфиденциальных данных или доступа к защищенным ресурсам.

VII. Заключение

В заключении можно отметить, что механизмы сервис-сервисной аутентификации в микросервисной архитектуре играют ключевую роль в обеспечении надежности взаимодействия компонентов приложения между собой. Разработчики должны учитывать требования к безопасности при проектировании механизмов взаимодействия между сервисами. При этом важно выбирать подходящие технологии и протоколы, а также тщательно тестировать систему на различных уровнях. Несмотря на некоторые сложности в реализации, микросервисная архитектура с механизмами взаимодействия между сервисами является эффективным подходом к разработке сложных

приложений, которые легко масштабировать и поддерживать в долгосрочной перспективе.

Представленная в статье архитектура может быть полезна не только для микросервисных приложений, но и для любых других приложений, особенно тех, которые требуют высокого уровня безопасности.

Таким образом, можно сделать вывод, что представленная в статье архитектура сервис-сервисной аутентификации является полезным инструментом для архитекторов микросервисных приложений и может быть применена в различных сферах, где требуется безопасная и эффективная межсервисная аутентификация.

Библиография

[1] Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L., 2017. Microservices: yesterday, today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195-216)

[2] Zheng, D. Y. (2018). A survey on security issues in services communication of Microservices-enabled fog applications. John Wiley & Sons, Ltd.

[3] Kai Jandera, Lars Braubachb, Alexander Pokahr. (2018). Defense-in-depth and Role Authentication for Microservice Systems. Procedia Computer Science, 456-463.

[4] Nacha Chondamrongkul, Jing Sun, Ian Warren. (2020). Automated Security Analysis for Microservice Architecture. IEEE International Conference on Software Architecture Companion.

[5] Peter Nkomo, Marijke Coetzee. (2019). Software Development Activities for Secure Microservices. B Computational Science and Its Applications – ICCSA 2019 (стр. 573-585). Springer Nature Switzerland AG 2019.

[6] Ali Rezaei Nasab, Mojtaba Shahin, Seyed Ali Hoseyni Raviz, Peng Liang, Amir Mashmool, Valentina Lenarduzzi. (2022). An empirical study of security practices for microservices systems. The Journal of Systems & Software.

[7] Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M., 2016. Microservice Architecture: Aligning Principles, Practices, and Culture. " O'Reilly Media, Inc.".

[8] Indrasiri, K. (2019). Microservice in Practice - Key Architectural Concepts of an MSA.

[9] Chandramouli, R. (б.д.). Security Strategies for Microservices-based Application Systems. 2019: NIST Special Publication 800-204.

[10] Soonhong Kwon, Sang-Jin Son, Yangseo Choi, Jong-Hyouk Lee. (2021). Protocol fuzzing to find security vulnerabilities of RabbitMQ. Special Issue: Convergence of cloud, Internet of Things, and big data: New platforms and applications (FiCloud2019). Transformative computing in security, big data analysis, and cloud computing applications (Transformative2020), Volume33, Issue23.

[11] Kamppuri, T. (2014). MESSAGE BROKERS AND RABBITMQ IN ACTION.

[12] Prabath Siriwardena and Nuwan Dias. (2020). Microservices Security in Action. New York : Manning Publications Co.

[13] Evan Gilman and Doug Barth. (2017). Zero Trust Networks: Building Secure Systems in Untrusted Networks. Sebastopol: O'Reilly Media, Inc.

[14] Siriwardena, P. (2014). Mutual Authentication with TLS. In P. Siriwardena, Advanced API Security (pp. 47-58). Maharagama, Sri Lanka: Apress Berkeley, CA.

[15] M. Jones, J. Bradley, N. Sakimura. (2015). RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force (IETF).

[16] Carlisle Adams, Steve Lloyd. (2003). Understanding PKI: Concepts, Standarts, and Deployment Considerations. Boston: Pearson Education, Inc.

Cross-Service Authentication Mechanisms in Applications with Microservice Architecture

K.I. Zimina, O.R. Laponina

Abstract **– In this article discussed cross-service authentication. It is one of the most important aspects of security in modern applications with microservice architecture.**

The basic mechanisms of service-to-service authentication, such as tokens and certificates usage, are represented in the article. An example of service-to-service authentication architecture is also given.

The article is a useful resource for developers who are working in the field of service-oriented architecture and interesting in security issues. It provides an extensive overview of authentication mechanisms and key points to consider during designing a microservice architecture.

The article proposes to use the mutual TLS (mTLS) protocol, which is the most popular way to secure cross-service communication during microservices deploying.

In this approach the responsible for the cross-service authentication lies with the mTLS proxy deployed for each microservice of the system. mTLS proxies work as intermediaries between the microservices, accepting requests for a secure communication channel establishment. The proxy approach simplifies the authentication process between two microservices, that can run on different platforms, by using different protocols and data formats.

By using the mTLS proxy, the solution is easy to scale, because it is enough to deploy a new instance of the mTLS proxy in case of new microservices appearing in the system. Also, the proxy does not depend on the language or system implementing an associated microservice, which makes the solution universal.

Keywords **– microservice architecture, authentication, trusted networks, mTLS, JWT, PKI.**

REFERENCES

[1] Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L., 2017. Microservices: yesterday, today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195-216)

[2] Zheng, D. Y. (2018). A survey on security issues in services communication of Microservices-enabled fog applications. John Wiley & Sons, Ltd.

[3] Kai Jandera, Lars Braubachb, Alexander Pokahr. (2018). Defense-in-depth and Role Authentication for Microservice Systems. Procedia Computer Science, 456-463.

[4] Nacha Chondamrongkul, Jing Sun, Ian Warren. (2020). Automated Security Analysis for Microservice Architecture. IEEE International Conference on Software Architecture Companion.

[5] Peter Nkomo, Marijke Coetzee. (2019). Software Development Activities for Secure Microservices. B Computational Science and Its Applications – ICCSA 2019 (стр. 573-585). Springer Nature Switzerland AG 2019.

[6] Ali Rezaei Nasab, Mojtaba Shahin, Seyed Ali Hoseyni Raviz, Peng Liang, Amir Mashmool, Valentina Lenarduzzi. (2022). An empirical study of security practices for microservices systems. The Journal of Systems & Software.

[7] Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M., 2016. Microservice Architecture: Aligning Principles, Practices, and Culture. " O'Reilly Media, Inc.".

[8] Indrasiri, K. (2019). Microservice in Practice - Key Architectural Concepts of an MSA.

[9] Chandramouli, R. (б.д.). Security Strategies for Microservices-based Application Systems. 2019: NIST Special Publication 800-204.

[10] Soonhong Kwon, Sang-Jin Son, Yangseo Choi, Jong-Hyouk Lee. (2021). Protocol fuzzing to find security vulnerabilities of RabbitMQ. Special Issue: Convergence of cloud, Internet of Things, and big data: New platforms and applications (FiCloud2019). Transformative computing in security, big data analysis, and cloud computing applications (Transformative2020), Volume33, Issue23.

[11] Kamppuri, T. (2014). MESSAGE BROKERS AND RABBITMQ IN ACTION.

[12] Prabath Siriwardena and Nuwan Dias. (2020). Microservices Security in Action. New York : Manning Publications Co.

[13] Evan Gilman and Doug Barth. (2017). Zero Trust Networks: Building Secure Systems in Untrusted Networks. Sebastopol: O'Reilly Media, Inc.

[14] Siriwardena, P. (2014). Mutual Authentication with TLS. In P. Siriwardena, Advanced API Security (pp. 47-58). Maharagama, Sri Lanka: Apress Berkeley, CA.

[15] M. Jones, J. Bradley, N. Sakimura. (2015). RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force (IETF).

[16] Carlisle Adams, Steve Lloyd. (2003). Understanding PKI: Concepts, Standarts, and Deployment Considerations. Boston: Pearson Education, Inc.