**<Impact of performance on security: JWT Token>**

**Name: Neelakshi Soni**

**<11 April , 2024>**

# Content

Acknowledgment

# List of Figures

# List of Tables

Abstract………………………………………………………………………………………………………………………………….. 6

Introduction……………………………………………………………………………………………………………………………. 6

Related Method…………………………………………………………………………………………………………………….. 12

Methodology…………………………………………………………………………………………………………………………. 14

Design and Implementation ……………………………………………………………………………………………. 14

1. Experiment ……………………………………………………………………………………………………………………… 15
2. Result ……………………………………………………………………………………………………………………………………. 21
3. Evaluation………………………………………………………………………………………………………………………………. 24
4. Conclusion ……………………………………………………………………………………………………………………………… 26

Acknowledgment

I am very thankful to Prof. Asif Imran and Prof. Ahmad Hadeh for their invaluable patience and support throughout my thesis. I appreciate you for all your valuable knowledge, feedback, and time.

# List of Figures

[Figure 1: Microservice architecture](#_List_of_Figures)…………………………………………………………………

[Figure 2: Structure of JWT Token](#_List_of_Figures)…………………………………………………………………….

[Figure 3: Principal propagation via JWT Token](#_List_of_Figures)………………………………………………………..

[Figure 4: Class diagrams of different services](#_List_of_Figures)…………………………………………………………..

[Figure 5: Experiment setup: e-commerce operation](#_List_of_Figures)

# List of Tables

[Table 1: Open source tool with uses…………………………………………………………………………………………….4](#_List_of_Tables)

Table 2: Microservices APIs…………………………………………..

# Table 3: Concurrent request without any security

[Table 4: Concurrent request with JWT token security](#_List_of_Tables_1)

[Table 5: Sequentially request without any security features enabled.](#_List_of_Tables)

[Table 6: sequential request with JWT token.](#_List_of_Tables_1)

# Abstract

Microservices are gaining a lot of popularity among industry users. These services are collections of independent, small services to make a big application. Yet, the basic and strong fundamentals of microservices such as scalability, loose coupling, and automation, can further make them more vulnerable to security.

Our research focuses on using the JWT Token to enhance the security of microservices for authentication purposes, considering performance. We have designed and implemented a simple security framework using JWT Token, interconnecting 3 small services using a composite microservice architectural pattern. We tried to emulate the real-world situation by sending 1000 concurrent and sequential requests with security features disabled and with a JWT token, where different microservices are communicating with each other. Our findings show that the performance overhead of using JWT Token is 11%, which can be a valuable insight for practitioners.

Index Terms—Microservices, REST APIs, SOA, distributed systems, cloud, authentication, REST, JWT.

# Chapter 1: Introduction

Microservices: -

Microservices have come to the fore from the traditional monolithic architecture. In the initial days, there was much demand for desktop or laptop devices. But then a change occurred, and mobile users rapidly started increasing. The industry shifted from monolithic architecture to microservices architecture. It is one of the most widely used and modern methods in the software industry, in which software is broken into small chunks of services. These small services are known as “microservices”. Few scholars have defined microservices differently.

* Newman [1] defines, microservices are small, autonomous services that follow seven principles, including modeling services around business concepts, automating processes, hiding internal details, decentralizing control, isolating failures, and enabling independent deployment and observability.
* Lewis and Fowler [2] define, microservices are an approach to building a single application as a collection of small services, each running in its process and communicating through lightweight mechanisms, often using HTTP resource APIs. These services are organized around business capabilities and are independently deployable with minimal centralized management. They can be written in different programming languages and use various data storage technologies.

Each microservice communicates with each other using the REST (Representational State Transfer) protocol. REST APIs work on HTTP protocol. Each microservice can have its databases.

A diagram of a service

Description automatically generated

**Figure 1: Microservice architecture**

Advantages of microservices: -

* Loosely coupled – each service is being developed independently and scaled automatically. Therefore, it makes it easier to understand, maintain, and develop.
* Highly scalable- each service is independent and can have its databases. Therefore, it becomes highly scalable.
* Continuous Integration and Continuous Delivery – using Jenkins, Hudson, etc., it can be easily deployed and developed. It automates the process of deployment and testing also.
* Modularity: We can implement smaller APIs, which can be used for multiple purposes compared to big, single-purpose APIs.
* Encapsulation: to an extent, they are encapsulated very well, each service can be written in different programming languages.

Scalability, automation, and separation of concerns are the big reasons for all big companies such as Netflix, Amazon, Uber, eBay, Twitter, and Airbnb to use microservice architecture. Experts have predicted that by 2024, 95% of all new apps will Feature Microservices Architectures.

Challenges associated with microservices: -

* Authorization across multiple services can be complex and risky.
* Small services communicate with each other, and that’s why secure and authenticated communication is very important.
* Exposing APIs to third parties in microservices makes them very vulnerable to attacks such as API abuse, SQL injection, and cross-site request forgery (CSRF).
* Microservices communicate with each other over the network. That’s why they become more prone to eavesdropping and interception.
* Microservices are highly scalable, making them susceptible to resource exhaustion attacks.
* Microservices use REST APIs. These are at risk of eavesdropping (sniffing), identity spoofing, session hijacking, Denial of Service (DoS), and Man-in-the-Middle (MITM) etc.
* Microservices are deployed on the cloud. Therefore, it leaves very little room to do anything related to technical aspects to prevent attacks or disruption.

|  |  |  |
| --- | --- | --- |
| **Category** | **Key Considerations** | **Popular Open-Source Tools** |
| Access Control | * RBAC/ABAC * Identification & Authentication * Authorization | OpenIAM, Central Authentication Service (CAS), Open Identity Platform |
| API Security | * Utilize a centralized gateway * Isolate microservices and embed security for each service * Secure coding practices | JSON Web Tokens (JWTs), OAuth, APIClarity, Zed Attack proxy (ZAP) |
| Secure coding practices | * Refresh tokens * Input and URL validation * Combine OAuth and SSL for enhanced security Encryption | SonarQube, OWASP Juice Shop, Code Warrior, Metasploit |
| Encryption | * Use standardized encryption algorithms * Enforce mutual-TLS (MTLS) to encrypt communications between services   Secrets management | mTLS, AES crypt, VeraCrypt, DiskCryptor, HTTPS Everywhere |
| Secrets management | * Avoid hard-coding secrets into source code/configurations * Utilize Vaults to encrypt secrets | Akeyless Vault, Docker Secrets, Vault |
| Vulnerability scanning and threat modeling | * Scan all open-source dependencies * Enforce automated, continuous scanning and threat modeling | OWASP Threat Dragon, Threagile, OWASP ZAP, OpenVAS |
| Incident response | * Continuously fine-tune alerts Incident response * Leverage managed incident response channels * Conduct post-mortem analysis to prevent recurring problems * Document runbooks to have a systematic and automated response to security problems | GRR Rapid Response, AlienVault OSSIM (Open Source Security Information and Event Management), Sans Investigative Forensics Toolkit (SIFT) Workstation. |

**Table 1 Open source tools with their uses**

JWT Token: -

In our paper, we have implemented JWT (JSON Web Token) for authentication for services. JWT allows for secure authentication. When a user logs in, they receive a JWT token. This token contains information about the user and is signed by the server. When the user makes subsequent requests, they include this token, allowing the server to verify their identity without the need to send credentials (e.g., username and password) with every request. JWT tokens can have an expiration time. This adds a layer of security by ensuring that even if a token is intercepted, it will only be valid for a limited time. It is a stateless service; it does not need to store session information. It can be encrypted, but mainly it is encoded and signed.

JWT is represented by JSON objects between different parties. They are very easy to transmit between parties. They securely encode the information about a user or session. JWT typically consists of: -

* Header - base64Url-encoded is used to form the first part of the JWT, and it is JSON-encoded. We have used RS256 as a signing algorithm.
* Payload – contains the actual data, that is being carried by the token. It contains the userID, user information, session, permission, expiration time, etc.
* Signature: - the header and payload are combined using a private key. Then by using a specific algorithm, the resulting string is base64Url-encoded.

A diagram of a software code

Description automatically generated

**FIGURE 2: STRUCTURE OF JWT TOKEN**

We are using two certificates here; both certificates have been generated using OpenSSL: -

* servicecert.pem – It verifies the JWT token with the help of a public key certificate. This certificate contains the public key. It is used to validate the authenticity of JWT tokens between different parties.
* servicekey.pem: - It contains the private key. It is used to sign the JWT token that has been created and used to prove the authenticity of JWT. It is kept private and secure.

We are using “RSA Signature with SHA-256”. JWT token has many different standards such as RFC 7519 - JSON Web Token (JWT), RFC 7515 - JSON Web Signature (JWS), RFC 7516 - JSON Web Encryption (JWE), RFC 7517 - JSON Web Key (JWK), RFC 7518 - JSON Web Algorithms (JWA), OpenID Connect, OAuth 2.0 Bearer Token Usage etc.

REST APIs: -

In our paper, microservices are communicating with each other through RESTful APIs. REST APIs are HTTP. These are language-agnostic.

* REST APIs do not store the user session information, these are stateless, which further helps in reducing the surface area.
* It uses the HTTP methods, which makes it easier to implement any security practices.
* Encryption protocols such as SSL/TLS can be used to secure data in transit.
* It is easier to give specific access for any permission to a specific resource.

Microservices have a dependency on each other. To fulfill any task, they need to rely on other services. This creates a complex web of interaction. Even if there is a security issue in one service, it can impact all the other services. Monolithic, SOA, and distributed had their security issues. However, microservices bring their challenges. Every kind of architectural service comes with its trade-off in security. Every organization needs to consider these challenges and make the choices. In recent years, microservice-style architecture has been adopted in most applications at the industry level. Now, even the source code is being exposed through third-party Web APIs, which raises another level of security concerns. Microservices offer their opportunities and challenges. The fundamental of microservice is that it needs to be scalable and loosely coupled with it to be able to be adopted by industry users. While maintaining the security of microservices, these things need to be kept in mind. There is a rising concern related to the security of microservices. In the world of self-driving cars, automation, and security are one of the greatest challenges. And there is a dire need to address these concerns in microservice architecture. JWT token is one of the security mechanisms, that is used for authorization of user session. In our paper, we have conducted experiments using e-commerce microservice and implementing with JWT token.

# Chapter 2: Related Work

The literature review is an integral part of this research. It provides a detailed analysis of previous research studies. This study section will highlight some of the best-published research that has implemented JWT tokens or any other security feat. To further help this study, we have also included relevant literature surveys.

Yarygina [3] experimented using a fictitious microbank to test it against using JWT token and MTLS service. They developed a small prototype to develop trust between individual microservices. Their study also showed that there was a little extra cost of performance, due to security features implementation. However, they emphasized the implementation of following the best practices and making the code open source for more research.

There were few studies dealing with the security of microservices and there was not much consensus among them. Another study was conducted, in which the author raised the concern about the security of microservices due to their very nature of individualism. They found that OAuth 2.0, OpenID Connect, API Gateway, and JWT are used to mitigate the risk of attack in microservices. There are not many open-source technologies that use the research mechanism except the Spring Framework.[5]

Tran Florén, and Simon researched on finding the best security pattern in microservices. They showed a relationship between response time and security. The longer the response time, the better security it has. In microservice [6]. It was seen that in lower traffic, the microservices applied rigorous access control checks. He suggested horizontal scaling is one of the solutions to improve performance. In his findings, the security concerns have outweighed the performance. The security pattern and its implication will depend on, particularly in which context it was implemented.

In another study, He, Xiuyu, and Xudong Yang. studied different methods, with their pros and cons. The paper talked about how distributed session solutions are very difficult to implement and maintain. They are very similar to the traditional auth way. If we use an SSO server, it might create problems related to traffic and SPOF. [34]

In another study conducted by Venčkauskas, Algimantas, et al., the author used an access control method to enhance the security of microservices. They tested and validated it against the centralized and decentralized architectures of microservices. They found that using access control in decentralized microservices enhances security by distributing the responsibility of access control to multiple microservices. It also helps with easy management of permissions. It also keeps the data safe and prevents the attacks on microservices.[4]

For the security of any system, it’s important to build up the proper digital infrastructure. There are security solutions available such as a secure socket layer (SSL), a virtual private network (VPN), a network firewall, API key, an encryption method, API key. A case study was conducted on electronic health coaching (eCoach). The author created the prototype for sharing personal health data between users. They implemented a robust SSK solution. They used the web engineering security method. They found that SSK solution was efficiently able to secure the microservices from an attack. 100% accuracy was observed. It was also sustainable under heavy loads. They also compared their SSK solution with other Keycloak-based security, Spring-based security, and hybrid security solutions. But SSK showed only the best outcome.[32]

Skatteverket, the Swedish tax agency, uses a microservice-based architecture. It runs on Kubernetes. They interviewed the people who are currently working in Kubernetes and microservice or have experience related to that, separately. They identified vulnerabilities and they did a case study on how to prevent those intrusions. They concluded that. Service mesh technology can help in reducing the attacks to the largest number.[17]

Which one is better, monolithic or microservice architecture? A study was conducted by Blinowski, Grzegorz, Anna Ojdowska, and Adam Przybyłek, to observe which architecture pattern performs better. It was found that on a single machine, a monolith outweighs microservice architecture. When there is a heavy load, JAVA can outperform other languages. Vertical scaling costs less on cloud platforms like Azure Cloud than horizontal scaling. The performance of a system decreases after a certain number of instances of scaling. Scalability performance does not depend on the kind of functionality.[14]

Shafabakhsh, Benyamin, Robert Lagerström, and Simon Hacks studied microservices from a performance efficiency standpoint. They researched which IPC method to be used for synchronous and asynchronous types of communication. How the selection of the IPC method can make a difference in system availability. Their findings show that asynchronous is much better than synchronous when it comes to performance and system availability. However, it also depends on which specific situation or context it has been implemented.[36]

In another study, the author observed that OpenID Connect is the best protocol to be used for authentication and authorization. As it uses OAuth 2.0 for authorization. The system does not need to store a username and password in the database. It reduces the overhead of maintaining storage for this separately. It also helps in preventing intrusion attacks.[37]

According to one study, microservices have its advantages, but they come with its vulnerabilities. It decreases the effectiveness of logaggregation architecture. The attack surface of modern applications increases which run on microservice architecture. The use of microservice increases the traffic level. The author suggested that a machine learning algorithm can be implemented to monitor any threats or the process of threat detection or any security vulnerabilities can be automated.[38]

From all the papers, it can be concluded that the implementation of any specific security mechanism can depend on the contexts and kind of service we want to provide, keeping in mind the infrastructure we are using.

# Chapter 3: Methodology

The security concerns related to microservices cannot be dealt with in the standard way. Netflix MTLS (closed source) and Docker Swarm MTLS are two security practices, that are being used in Industry. However, the cost of using both services was quite unknown. In our paper, we have used a fictitious e-commerce service, whose performance we have evaluated without any security features in place and using JWT Token for authorization purposes.

1. Design and Implementation

We have implemented an e-commerce service and Reverse STS service, that is generating JWT tokens. JWT tokens are only generated after the username and password have been validated. What is the authorization state of the user, it needs to be known by microservice. Whenever a user interacts with any microservice, it needs to be validated. Even if one microservice calls another microservice, the JWT token will be validated again multiple times. If the validation of JWT fails, it will decline the request or send an error.

Token-based authentication: -

It is a very commonly used security mechanism. It uses cryptographic objects. These objects are known as security tokens. These tokens are generated using certain algorithms. These tokens contain information needed for authentication or authorization information. These tokens are generated on the server side and passed onto the client. It has an expiry time. It is very scalable, as it requires only user information. These tokens can be generated using the role-based approach, attribute-based approach, and fine-grained approach. HTTP cookies are a very well-known example of token-based authorization.

In our paper, Reverse STS service, we have three caveats.

* The clock synchronization problem: - does not exist. This is an assumption. The issuing node and validating nodes both have their clocks synchronized.
* Protected channel: - we are assuming that the tokens are passing through a secure channel like TLS. If the channel is not safe, it can be intercepted and reused within its expiry time. To make it safer, we can shorten the lifespan of tokens.
* Private key – it is assumed that it remains safe all the time. An attacker can impersonate the key, if not kept safe.

Diagram of a diagram of a system

Description automatically generated

**Figure 3: Principal propagation via JWT Token**

A generic token-based authentication scheme for microservices that enables user-to-service authentication and identity propagation based on cryptographic tokens. 1) Incoming user requests first go to API Gateway. 2) ApI-Gateway redirects it to the user authentication service for the generation of a security token. 3) Requesting a security token. If the user is authenticated successfully, an access token is generated using user information. 4) Returning a security token. The token is being validated and passed along if it is valid to access any microservice. This token is designated for internal use only, is being kept on the server side, and is not given to the user.

B. Experiment

Security is not measured with performance. However, it’s been a vital factor in adopting or rejecting any security practices in the industry. In our paper, we have addressed this issue, how a common security mechanism can impact the performance of microservices.

1)E-commerce

We have developed a fictitious e-commerce application, which consists of the following micro-services.

1. User Service – It manages the users. It has the functionality to add users, remove users, and log in to a user. Users can log in with the help of a username and password. The user will be issued a UserID and JWT token, which will have an expiry time.
2. Account Service – every user will have an account linked with userID. This service has the functionality of getting account numbers related to that particular user ID, and updating the balance of a particular account no or account not associated with the userID . Users can close the account, open the account, and update the balance in the account through userID and account number.
3. Inventory Service – It has an inventory of 1000 products with product, product description, product price, and stock. It has the functionality of adding any new product with all its details, to get all the details through product ID and updating of stock through the productid.
4. Cart Service – every user has his own cart. Users can add the items to the cart, remove them from the cart, and check out the cart.
5. API gateway: The main entry point to the system.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Figure 4 : Class diagrams of different services**

The cart service communicates with the inventory service and account service. To check out the cart, it will first get the details of the product, then check the available balance by communicating with the account service.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Microservice** | **URL** | **HTTP Method** | **Input** | **Function** |
| 4 microservices | / | GET | None | Get the microservice |
| /{id} | GET | Microservice ID | Get the microservice |
| / | POST | New microservice | Create a microservice |
| /{id} | PUT | Existing info of microservice | Update the microservice, apply to microservices |

**Table 2: Microservices APIs**

The system is written in the Python v3.1 programming language. All the HTTP methods get, put, delete and update are accessed using REST APIS. We have implemented all of these using Flask. We can set the number of workers per server. Account and user service maintain their individual SQLite databases, making every microservice loosely coupled. To maintain the simplicity of the experiment, each microservice has only one instance. Each service can run individually. We have run it on Windows 10 64-bit operating system.

Code is written in such a way that you can add multiple users and concurrently, they can perform operation as well as sequentially. If the test client wants to increase the number of CPUs, then the code also works for multiprocessing. All the services follow a “Shared-nothing” architecture. All services have their database and resources. Our e-commerce service also follows a composite architectural pattern.

A diagram of a service

Description automatically generated

**Figure 5. Experiment setup: e-commerce operation**

The test client registers two users and opens accounts for them. Consecutively, a balance is added to their accounts. Then, the test client is tested against 1, 10, 100, 500, and 1000 requests. These requests are sent to both users concurrently and subsequently. To perform checking out operations. We add the number of requested products to the respective user’s cart and, after that, carry out a series of checkout operations. For every request, the experiment is performed without any security features or by using a JWT token for authorization. To measure the system performance, an average execution time of checkout requests is calculated on the client side. Total response time and throughput are also noted for checking out the cart for all number of requests. The checkout operation involves four microservices as shown in Figure 5. Several factors contribute to the time it takes to perform one payment operation. These factors are network delays and processing time inside microservices including database access time. In the given setup, checkout operations always succeed. If a user’s account does not have enough balance, then it fails.

We ran the experiment in two parts:

• Baseline. Running the test client against the e-commerce model with security features disabled.

* + 1. Subsequential requests
    2. Concurrent requests

• Tokens. Running the test client against the bank model with the Reverse STS service in place and JWT tokens validation.

* + 1. Subsequential requests
    2. Concurrent requests

The experiment was run on a Microsoft Pro with 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz and 8.00 GB (7.83 GB usable) of memory. It was run via a local host, and both the test client and e-commerce model were run on the same machine.

The same pattern is observed for sequential requests. However, the overhead introduced by the JWT token remains consistent among all the requests. When JWT security is applied, it takes an additional time to process each request for validation of the token, which further consumes more resources and time.

# Chapter 4: Analysis of the Results

Our experimental research aimed at evaluating the JWT token access method on various criteria- CPU load, RAM usage, and query execution time. We have sent both concurrent and sequential requests to two users. The code is written in such a way that the number of users and number of CPUs can be increased.

Before sending any request, CPU utilization was 0%.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Total no of requests** | **CPU usage** | **Memory usage** | **Total Response Time(seconds)** | **Throughput (Transactions per second)** |
| 1 | 0.1% | 16.3% | 1.004 | 0.99194 |
| 10 | 0.1% | 16.3% | 1.02808 | 9.7268 |
| 50 | 0.5% | 16.3% | 1.0838 | 46.130611 |
| 100 | 1.0% | 16.2% | 1.177 | 84.894 |
| 500 | 2.7% | 16.3% | 3.65 | 136.95 |
| 1000 | 3.2% | 16.3% | 5.23 | 191.11 |

**Table 3: Concurrent request without any security**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Total no of requests** | **CPU usage** | **Memory usage** | **Total Response Time** | **Throughput (Transactions per second )** |
| 1 | 3.5% | 23.5% | 0.016 | 61.38 |
| 10 | 3.6% | 23.5% | 0.05 | 177.57 |
| 50 | 3.5% | 23.4% | 0.223 | 223.551 |
| 100 | 3.5% | 23.4% | 0.4628 | 216.131 |
| 500 | 3.5% | 23.4% | 2.514 | 198.83 |
| 1000 | 3.5% | 23.4% | 5.05 | 197.92 |

**Table 4: Concurrent request with JWT token security**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Total no of requests** | **CPU usage** | **Memory usage** | **Total Response Time(seconds)** | **Throughput (Transactions per second)** |
| 1 | 1.4% | 16.2% | 0.010 | 100 |
| 10 | 1.9% | 16.2% | 0.03 | 333 |
| 50 | 6.5% | 16.2% | 0.124 | 403.06 |
| 100 | 6.5% | 16.2% | 0.24 | 416 |
| 500 | 6.5% | 16.2% | 1.51 | 331 |
| 1000 | 6.7% | 16.2% | 2.127 | 470 |
| 2000 | 7.1% | 16.9% | 4.66 | 429 |
| 5000 | 7.5% | 16.9% | 11.23 | 445 |
| 20000 | 8.3% | 20.0% | 19.16 | 1043 |

**Table 5: Sequentially request without any security features enabled.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Total no of requests** | **CPU usage** | **Memory usage** | **Total Response Time** | **Throughput (Transactions per second )** |
| 1 | 1.7% | 23.8% | 0.010 | 100 |
| 10 | 1.9% | 23.8% | 0.031 | 322 |
| 50 | 7.6% | 23.7% | 0.130 | 384 |
| 100 | 7.7% | 23.6% | 0.27 | 370 |
| 500 | 7.6% | 23.6% | 1.55 | 322 |
| 1000 | 7.7% | 23.6% | 2.47 | 404 |
|  |  |  |  |  |

**Table 6: sequential request with JWT token.**

From Table5 and Table 6 , it is evident that as the number of requests increases, the latency incareses. However , while using JWT token for sequential request, we observed 5% overhead of using JWT token for sequential request. The CPU udauges and memory usuages also incarese with the increase in number of rqeuests. For lower number of request , the latency remains almost same for with using JWT token and without JWT. As the number of request increases, we observed a relatively small incarese in latency. CPU usage and memory usage also incareses with JWT token.

Same patten was observed for concurrent request.

Figure 6 represents the result of our experiment. As expected, the throughput is much higher in the case of sequential requests without any security. A closer inspection of the data shows that performance, for concurrent requests, for a single request, the overhead with the JWT token is 76.92%, which has decreased to 41.41% for 1000 requests. It shows that when the system handles more concurrent requests, the overhead introduced by the JWT token decreases. CPU usage was intensive for more requests, however, the memory consumption remains fairly the same for each request.

The same pattern is observed for concurrent requests. However, the overhead introduced by the JWT token remains consistent among all the requests. When JWT security is applied, it takes an additional time to process each request for validation of the token, which further consumes more resources and time.

It is observed that concurrent processing resulted in higher latency compared to sequential processing, albeit slightly higher latency due to JWT token security measures.

“””

Across all the tables, as the payload increases, the latency tends to increase as well. This is due to the increased processing time of all requests, which results in longer response time for individuals. The latency may increase linearly with request volume, after a point it might reach a plateau for higher number of requests.

For concurrent number of requests, the throughput might increase initially as resources are fully utilized. However, as the number of requests incareses, might decrease due to resource contention.

Using JWT token increases CPU usages and memory usages slightly. This is because JWT token validation and generation involves additional cryptographic operations. However, the effect on total response time and throughput varies depends on specific payload.

Sequential request ahs exhibited lower CPU usages and memory usuges. This can be due to para…………………..

Yarygina [3], in their paper presented their result of using JWT token. In their paper, they have observed 7% overhead using JWT , while in our experiment we have observed 13% overhead. Same as our experiment result, the difference between the baseline case and case with security mechanisms is in place is relatively small. Microservices are generally slow, however for security critical application this overhead is still acceptable. Using JWT token comes with, high cost of setting up HTTPS connections an d validation of tokens, which can be reduced by using caching, pooling and reusing existing connections. In Figure 8 , I have shown the comparison bweteen our experiment and Yarygina’s experiment.

Ueda et.al. [39], in their paper, experimented and found that, on the same hardware configuration, the performance of a microservice can be 80% lower than the monolithic version. The overhead is significantly too high, and the industry is willing to take it, which shows that the impact of security mechanisms becomes insignificant in comparison.

The relatively slow performance of our e-commerce model can be attributed to the following:

1) Synchronous HTTP communication: sequential requests and no parallelization.

2) Slow database access.

3) Python is not as efficient as certain other programming languages.

4) Suboptimal configuration of the web servers. Also, the e-commerce model is only a proof-of-concept and offers limited performance and no scalability.

**Evaluation**

1) **Security Considerations:**

We have used well-known security mechanisms such as JWT. No new protocols related to cryptography were proposed. There might be implementation flaws as well. ReverseSTS service is very critical from a security point. To get a more accurate and better result, we need to run it in a more rigorous environment. Token authentication service can be implemented with access control, attribute control, or any customized security feature, where we can grant some permission or access to a user, or maybe put more information or longer messages in the token to make it more secure. We have used only 1 CPU; more CPUs can be utilized for load testing using any cloud provider's service is one of the solutions. Such potential future studies could either use JWT alternatives or use something entirely different, such as cryptographic keys or XACML. To investigate further, these different technologies can be implemented and compared in terms of resource consumption and scalability.

2) **Performance Evaluation:**

It has been observed that as the number of requests increases, our experiments show that a microservice network introduces latencies in the order of milliseconds; in this setting the performance hit of basic security features becomes negligible. More rigorous and realistic load-testing can be done to make the CPU perform under intensive tasks and observe how the system reacts to it. To reduce the overall response time from the system, caching could be implemented at the API gateway or at the user authentication service. As soon as the token expires, it needs to be removed by implementing a cache eviction policy. At every microservice, we validate the JWT token. It can be implemented with only a few relevant services. If we have more number of requests per second, to reduce overhead, multiple internal API gateways can be implemented. A unique request identifier can be added to the JWT token request. That can keep the trace of activity logs of the user. It can give an idea of which specific functions have been performed. Different algorithms for the signature of JWT tokens can be tried out, which have high speed and better performance. The Ed448 (448 bits) and Ed25519 (255 bits) versions of the elliptic curve signature algorithm are good in speed. To curtail the number of requests or prevent any DOS attack from one user or any device Id or any same IP address, or from the same credentials, a rate limiter or load balancer can be put at API Gateway.

3) **Framework Limitations:**

We have just tested this on 1 security feature, many important security features are missing. The test client can be tested against multiple security features. The framework is too simple and easy to implement. There are no key rotation or key revocation mechanisms. We have addressed authentication, not authorization. In token-based authentication, we can grant certain permissions or rights to users, to access only a few resources or microservices. We have tested it using Python language. It can be tested against using other languages. We have not created replicas of databases; therefore, the system becomes more vulnerable to attacks, as it creates the problem of a single point of failure. It needs to be hosted on the cloud.

4) **Reproducibility and Future Work:**

Every research has its limitations. We conducted our experiment with limited resources. We have used a limited number of microservices with few transactions, it might be possible that this approach does not go well with other applications. Further testing and research need to be done to fully test the proposed methods in different scenarios. There are many other securities risks, that the proposed token-based approach might not be able to reduce. Therefore, the use of the JWT token can enhance security, but it may not be able to provide a holistic approach to the system.

The source code for e-commerce and ReverseSTS can be found on my GitHub(https://github.com/NeelakshiSoni007/Thesis\_new ). We intend to improve on above mentioned limitations for future research.

# Chapter 5: Conclusion

In this paper, we have implemented JWT tokens with a focus on security. Abstraction, reusability, and separation of concerns are basic concepts of microservices. These concepts bring new challenges to security. In our case study, we experimented with a JWT token and without any security.

To ensure only authorized clients can access, we can implement external and internal token mechanisms. By using the JWT token, the risk of cross-site scripting (XSS) and SQL injection can be reduced by verifying the user information with the secret key. The access can be denied if client-side token information has been tampered with. This approach not only can help in securing sensitive data and resources, but also reduce the risk of attacks.

By promoting a secure culture within an organization, trust can be built between stakeholders and customers. Our finding shows increased overhead, due to the validation of JWT token at each microservice access. By hosting it on the cloud, and using cloud resources such as load balancer, and rate limiter, the performance overhead can be reduced significantly.

With the increased adoption of microservices at the industry level, the best practices need to be followed to secure communication at microservices. These security solutions need to be easy to implement, accessible to the world, and efficient in terms of performance.

# Chapter 6: Bibliography

[1] S. Newman, Building Microservices. O’Reilly Media, 2015.

[2] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, “Microservices in practice (part 1): Reality check and service design,” IEEE Software, vol. 34, pp. 91–98, January-February 2017.

[3] Yarygina, Tetiana, and Anya Helene Bagge. "Overcoming security challenges in microservice architectures." *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE, 2018.

[4] Venčkauskas, Algimantas, et al. "Enhancing Microservices Security with Token-Based Access Control Method." *Sensors* 23.6 (2023): 3363.

[5] Góes de Almeida, M., & Canedo, E. D. (2022, March 16). Authentication and Authorization in Microservices Architecture: A Systematic Literature Review.

[6] Tran Florén, Simon. "Implementation and Analysis of Authentication and Authorization Methods in a Microservice Architecture: A Comparison Between Microservice Security Design Patterns for Authentication and Authorization Flows." (2021).

[7] C. Otterstad and T. Yarygina, Low-Level Exploitation Mitigation by Diverse Microservices. Springer, 2017, pp. 49–56.

[8] Y. Sun, S. Nanda, and T. Jaeger, “Security-as-a-service for microservices-based cloud applications,” in Cloud Computing Technology and Science (CloudCom 2015). IEEE, 2015, pp. 50–57. [

9] S. Kim, F. B. Bastani, I.-L. Yen, and R. Chen, “High-assurance synthesis of security services from basic microservices,” in Software Reliability Engineering (ISSRE 2003). IEEE, 2003, pp. 154–165.

[10] N. Josuttis, SOA in Practice: The Art of Distributed System Design. O’Reilly Media, Inc., 2007.

[11] M. Richards, Microservices vs. Service-Oriented Architecture. O’Reilly Media, 2015.

[12] O. Zimmermann, “Microservices tenets: Agile approach to service development and deployment,” Computer Science - Research and Development, pp. 1–10, 2016.

[13] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, “Microservices in practice, part 2: Service integration and sustainability,” IEEE Software, vol. 34, no. 2, pp. 97–104, Mar. 2017.

[14] A. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigms. Pearson Prentice Hall, 2007.

[15] A. Madhavapeddy and D. J. Scott, “Unikernels: Rise of the virtual library operating system,” Queue, vol. 11, no. 11, pp. 30:30–30:44, Dec. 2013.

[16] J. Armstrong, “Making reliable distributed systems in the presence of software errors,” Ph.D. dissertation, The Royal Institute of Technology, Stockholm, Sweden, Dec. 2003.

[17] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, “Flip feng shui: Hammering a needle in the software stack,” in 25th USENIX Security Symposium. USENIX, Aug. 2016, pp. 1–18.

[18] V. Mavroudis, A. Cerulli, P. Svenda, D. Cvrcek, D. Klinec, and G. Danezis, “A touch of evil: High-assurance cryptographic hardware from untrusted components,” in Computer and Communications Security (CCS’2017). ACM, 2017, pp. 1583–1600.

[19] T. Combe, A. Martin, and R. Di Pietro, “To Docker or not to Docker: A security perspective,” IEEE Cloud Computing, vol. 3, no. 5, pp. 54–62, 2016.

[20] H. Takabi, J. B. D. Joshi, and G. J. Ahn, “Security and privacy challenges in cloud computing environments,” IEEE Security Privacy, vol. 8, no. 6, pp. 24–31, Nov 2010.

[21] C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful web services vs. Big Web Services: Making the right architectural decision,” in 17th Int’l World Wide Web Conf. (WWW2008), Beijing, China, 2008, pp. 805–814.

[22] T. Yarygina, “RESTful is not secure,” in Applications and Techniques in Information Security (ATIS 2017). Springer, 2017, pp. 141–153.

[23] (2014, Apr) The Heartbleed bug in OpenSSL library. [Accessed Nov. 1, 2017]. <http://heartbleed.com/>

[24] B. Moller ¨, T. Duong, and K. Kotowicz. (2014, Sep) This POODLE bites: Exploiting the SSL 3.0 fallback. [Accessed Nov. 1, 2017]. https://www.openssl.org/∼bodo/ssl-poodle.pdf

[25] (2017) Owasp top 10 – 2017: The ten most critical web application security risks. [Accessed Nov. 1, 2017]. https://www.owasp.org/ images/7/72/OWASP Top 10-2017 %28en%29.pdf.pdf

[26] F. Montesi and J. Weber, “Circuit Breakers, Discovery, and API Gateways in Microservices,” CoRR, vol. abs/1609.05830, 2016.

[27] C. Richardson and F. Smith, Microservices from Design to Deployment. NGINX, Inc., 2016.

[28] Y. Yarom and K. Falkner, “Flush+reload: A high resolution, low noise, l3 cache side-channel attack,” in 23rd USENIX Security Symposium. USENIX, 2014, pp. 719–732.

[29] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre attacks: Exploiting speculative execution,” ArXiv e-prints, Jan. 2018.

[30] M. Coblenz, J. Sunshine, J. Aldrich, B. Myers, S. Weber, and F. Shull, “Exploring language support for immutability,” in Int’l Conf. on Software Engineering, ser. ICSE ’16. ACM, 2016, pp. 736–747.

[31] D. Monica. (BSides Lisbon 2016) MTLS in a microservices world. ´ [Accessed Nov. 1, 2017].

[32] Chatterjee, Ayan, and Andreas Prinz. "Applying spring security framework with keycloak-based oauth2 to protect microservice architecture APIs: A case study." *Sensors* 22.5 (2022): 1703.

[33] Muresu, Daniel. "Investigating the security of a microservices architecture: A case study on microservice and Kubernetes Security." (2021).

[34] He, Xiuyu, and Xudong Yang. "Authentication and authorization of end user in microservice architecture." *Journal of Physics: Conference Series*. Vol. 910. No. 1. IOP Publishing, 2017.

[35] Blinowski, Grzegorz, Anna Ojdowska, and Adam Przybyłek. "Monolithic vs. microservice architecture: A performance and scalability evaluation." *IEEE Access* 10 (2022): 20357-20374.

[36] Shafabakhsh, Benyamin, Robert Lagerström, and Simon Hacks. "Evaluating the Impact of Inter-Process Communication in Microservice Architectures." *QuASoQ@ APSEC*. 2020.

[37] Shaikh, S., & Mane, S. B. (2017). Authentic techniques of authentication in microservices. International Journal of Current Advanced Research, 6(4), 3342-3345. DOI: 10.24327/ijcar.2017.3345.0267.

[38] Barabanov, A., & Makrushin, D. (2020). Authentication and Authorization in Microservice-Based Systems: Survey of Architecture Patterns. DOI: 10.21681/2311-3456-2020-04-32-43.

[39] T. Ueda, T. Nakaike, and M. Ohara, “Workload characterization for microservices,” in IISWC 2016. IEEE, 2016, pp. 85–94.