International Journal of Current Advanced Research
ISSN: O: 2319-6475, ISSN: P: 2319 – 6505, Impact Factor: SJIF: 5.995
Available Online at www.journalijcar.org
Volume 6; Issue 4; April 2017; Page No. 3342-3345
DOI: http://dx.doi.org/10.24327/ijcar.2017.3345.0267

Research Article

AUTHENTIC TECHNIQUES OF AUTHENTICATION IN MICROSERVICES

Shagufta N. Shaikh[1] and Sunil B. Mane[2]

[1]Department of Computer Engineering, College of Engineering Pune, Pune – 411005
[2]Department of Computer Engineering & IT, College of Engineering Pune, Pune – 411005

A R T I C L E I N F O

Article History:

Received 19th January, 2017
Received in revised form 10th February, 2017
Accepted 22nd March, 2017
Published online 28th April, 2017

Key words:

Microservices, OAuth 2.0, OpenID Connect

A B S T R A C T

Microservices is the catch word of the town nowadays. The microservices are small, autonomous services doing a single task, and performing it well. However various concerns such as security in microservices are not explored yet. This paper presents a comparison of the existing protocols such as 2-way SSL, HMAC, SAML, etc. used for authentication and authorization of the end users by the service providers. It also explores the concerns where they lack and presents a model implementing OpenID Connect. It presents a proper comparison to propose OpenID Connect to be best of the lot.

INTRODUCTION

The rudimentary approach for developing software has been the monolithic way. Monolithic approach is still good for small scale teams and projects, nevertheless once scalability, flexibility and other requirements like fast development, short time to market, wider team alliance, and so on becomes gradually critical to accomplish business competitiveness, monolithic halts being profitable. This is where the Microservices architecture comes to rescue. Microservices is responsible for an intensive, scoped and modular tactic for application design. Microservices are small, autonomous services that work together. [1] It can be well elaborated using keywords: 'Faster development and Speed to production'. Microservices are deceptively termed to be code of limited length. Conversely, microservices are a piece of code which performs a single task and performs it soundly. They are independent in failure i.e. failure of a single component does not force the entire system to breakdown at once. The term micro indicates the services to be lightweight and which cannot be further divided into sub tasks and performs one task solely with minimal dependency on other services. They are independently scalable as well. The most perplexing part of microservices is defining the granularity of the services.

Security in microservices is one of the least explored topics. This paper explores the various vulnerabilities in security and also presents the various methods deployed for providing authentication and authorization in microservices. Since microservices depends on the idea of loose coupling and high cohesion. They do not share any databases. If at all there are any dependencies among the microservices they use light weight communication mediums to achieve it. The most common and widely used communication methodology today are the REST APIs. REST APIs are simple, stateless and lightweight protocol used for communication. As REST: Representational State Transfer is stateless several traditional authentication and authorization techniques fail to suffice the purpose. Several protocols are being modulated for securing the REST APIs. However there isn't a standard protocol for securing microservices. This paper provides a crisp comparison of the several traditional and upcoming techniques for providing authentication as well as authorization in microservices. It also implements a model on the OAuth 2.0 and OpenID Connect techniques employed for the authentication and authorization in microservices.

The further sections of the paper is as follows: Section 2 briefs about the topics that gave motivation for this paper. Section 3 explains the various perspectives involved related to security in microservices. Section 4 describes the various traditional proposed solution for authentication and authorization in microservices. Section 5 presents the implemented model of the paper presenting OAuth2.0 and OpenID Connect techniques. Section 6 provides a crisp comparison of the strengths and flaws of the developed techniques. Section 7 puts forth the accomplishments of the paper.

LITERATURE SURVEY

Microservices has become a hot topic in field of software development. Its efficiency is well demonstrated by big giants like: Amazon, Netflix, eBay, etc. The book [1] on

Corresponding author:* **Shagufta N. Shaikh
Department of Computer Engineering, College of Engineering Pune, Pune – 411005

microservices can be well called the fundamental guide to developing microservices. It exposes the concepts related to data partition, service discovery, circuit breakers, etc. that are required to be kept in mind while developing microservices. Eric Evans [3] describes the methodology that is to be adapted for modularizing the domain. It highly encourages designing of applications on the tunes of "loose coupling and high cohesion".

Alshuqayran et.al. in their paper [2] identifies and presents various architectural challenges related to microservice systems. This paper emphasizes on the fact that security in microservices is the least explored topic while designing and developing the microservices. It diagrammatically explains that only about 9% of the research is carried on microservice security as compared to the other concerns. This motivates to deep dive into the arena of security for microservices. Security is a gigantic topic, and thus can't be covered in a single paper. This paper focuses on the authentication and authorization module of security in microservices. It attempts to formulate a standardized model for authentication and authorization in microservices.

Various Standpoints in Securing Microservices

Microservices are vulnerable to several security issues. The security of microservices can be visualized in a number of standpoints. They are as follows:

Safe development lifespan and Test Automation

The backbone strategy of the microservices is the pace of development. The microservice should be simplified and quick to develop, scale, alter, test and deploy. While developing microservices, the various security vulnerabilities must be considered right from planning and designing stage. This facilitates a robust microservices to withstand several outbreaks.

DevOps Security

Microservices have various deployment configurations, the widely adopted pattern is one service per host. The host is usually a container (e.g. Docker). The containers by default have no security employed. Thus making our services more vulnerable to attacks. The alternatives for secluding the containers and the granularity to which it must be secluded must be planned.

Application Security

This is mainly considered with the access control of the users to the deployed microservices. The application as a whole is not exposed to the users. Only few microservices of an application gets direct exposure to external world. Hence the security concern can be converged only on such microservices. The challenge is to authenticate the consumer and permit the login context amongst the microservices, in a symmetric routine, thus consenting each microservice to approve the user.

Theoretical Background

This section takes into account the various traditional techniques that were employed for providing authentication and authorization among the services. It also pin-points the features due to which these techniques do not suffice the requirements of microservices.

2-way SSL / TLS

SSL/TLS scheme is only used for authentication. Furthermore it requires server to store clients' certificate thus violating the stateless property of REST used for communication among the microservices.

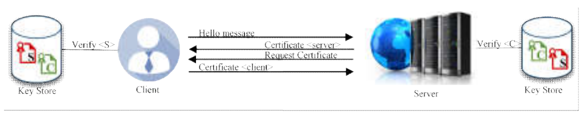

Figure 1 2-Way SSL / TLS Flow

HMAC Signing

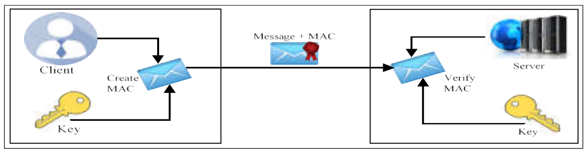

Figure 2 HMAC Flow

HMAC signing is too used for authentication only. It also requires sharing of keys thus weakening the solution. For each request the MAC will differ. Thus requires the entire procedure to be executed every single time.

SAML

SAML is very efficient mechanism but with SOAP- a predecessor of REST APIs. Here the users log in into the Identity provider and obtain a SAML. It then uses this SAML with the service providers to access their services. This protocol only makes use of one representation i.e. XML. However in modern day Json etc. representation have gained wide popularity. Also once the identity provider sends an assertion it deletes it from its database. Hence the service provider has no means to reassure itself later if required.

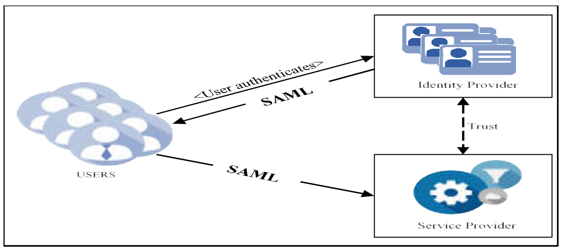

Figure 3 SAML Flow

Proposed Security

The microservice require a robust authentication and authorization mechanism as it is highly distributed in nature. The most widely accepted protocol is OAuth 2.0. However there is an upcoming protocol OpenID Connect for both authentication and authorization. Both these protocols does not require the server to maintain sessions. Thus withstanding the REST and microservices constraint of stateless servers. Furthermore they eliminate the time and labor required for creating accounts for every service to access secured resources. They also do not compromise the privacy of the users. The users are relieved of entering their private information for registering with the several applications. They can simply exchange tokens and get themselves validated by the services. This paper implements both the protocols, and presents a comparison among them.

OAUTH 2.0

The figure is self-illustrative of the flow of events in OAuth 2.0. The detailed descriptions of OAuth 2.0 terminologies are as follows:

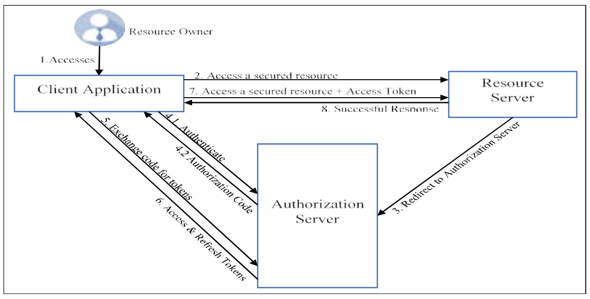
Resource Owner
1.Accesses
Client Application
2. Access a secured resource
7. Access a secured resource + Access Token
8. Successful Response
Resource Server
4.1 Authenticate
4.2 Authorization Code
5. Exchange code for tokens
6. Access & Refresh Tokens
Authorization Server
3. Redirect to Authorization Server

Figure 2 OAUTH 2.0 Flow

Table 1 OAuth2.0 Terminologies

Actors	**Clients, Authorization Servers, Resource Server, Resource owner**
Scopes	Permissions
Tokens	Access tokens, Refresh Tokens
Passing tokens	By value, by reference (*does not define Token format*)
Profiles of tokens	Bearer, Holder of Key
Types of tokens	WS-Security, SAML, JWT, Custom

Problem

OAuth2.0 though a popular protocol, has a lot of controversies associated with it. The first and foremost is that it brings lack of anonymity. When one logs in via an authorization server it gives the application rights or authorization to view its personal details stored there. OAuth is safe only when implemented correctly. However it does guarantee that this process is full proof. It can lead to security attacks like phishing, where innocent users may be prompted by a look a like authorization server and asked to enter their credentials. Thus leading to severe crimes.

OAuth is not a substitute for login functionality. It is developed to be used in scenarios where one requires to import its data from Website A to Website B. The very misconception in the use of OAuth2.0 is that it suffices both the needs of the application i.e. Authentication along with Authorization. It in fact does none. Furthermore even if OAuth relieves users from inflowing passwords for several websites it does not provide complete security. Because though the passwords won’t be intercepted but in future if the Resource Owner gets compromised the authorization that one provides can be exploited. For e.g., if one allows a website A (resource owner) to post on the Facebook page with the use of OAuth, and in future Website A gets hacked. The hacker will now be provided with the luxury of posting anything he desires on one’s Facebook page with the help of the OAuth bearer tokens and permissions one had granted.

The use of bearer tokens is another threat. It is not secured can lead to replay attacks. For example: Use of Authorization Token can be imagined as the use of Cash. Once the cash comes in hand of another person it can easily use it. In analogy once the bearer token is acquired one can easily impersonate the owner and access the secured resources from the resource owner. OAuth is more about delegation. It tells the resource owner that the authorization server does recognize the user. But it has no means for the resource owner to determine the legitimate owner. This leads to a major security concern. This is where the OpenID Connect plays a vital role. It is well elaborated in section 5.2.

OpenID Connect

OpenID Connect a successor of OpenID, is an identity layer over the OAuth2.0 protocol. It is a guideline that put in order how an identity provider and trusting associates can use OAuth2.0 to communicate identity data to one another. It allows the application to verify the owner directly. It is simple, interoperable, flexible and secure. It is better fit for microservices. As it provides both identity token along with authorization token in one request. It standardizes the security protocol. It allows Clients to verify the identity of the resource owner based on the authentication achieved by an Authorization Server, as well as to attain elementary profile evidence about the resource owner in an interoperable and REST-like style. The specification set is extensible, allowing participants to use voluntary features such as encryption of identity data, discovery of OpenID Providers, and session management, as desired. Also the use of JWT tokens makes it more secure, as these tokens are encrypted and implemented by applying HMAC over them. Thus attacks like replay attacks, impersonation, etc. gets eliminated.

Thus OpenID Connect would be the optimal protocol for all sorts of cloud computing technologies as it fulfils nearly all of the requirements.

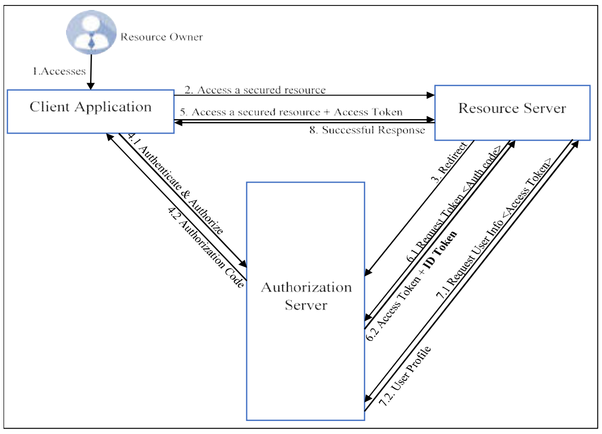
Resource Owner
1.Accesses
Client Application
2. Access a secured resource
5. Access a secured resource + Access Token
8. Successful Response
Resource Server
4.1 Authenticate & Authorize
4.2 Authorization Code
Authorization Server
3. Redirect
6.1 Request Token <Auth code>
6.2 Access Token + ID Token
7.1 Request User Info <Access Token>
7.2. User Profile

Figure 5 OpenID Connect Flow

Implementation Outcomes

This paper proposed an implementation of both OAuth 2.0 and OpenID Connect. The technology stack consisted of Spring Tool Suite as the IDE for developing Java-based applications. The graphs listed below is captured using jvisualVM. It is used to graphically observe, troubleshoot, and sketch Java applications. It is available with the jdk-kit itself.

The memory usage profiles of both the protocols are attached as below:

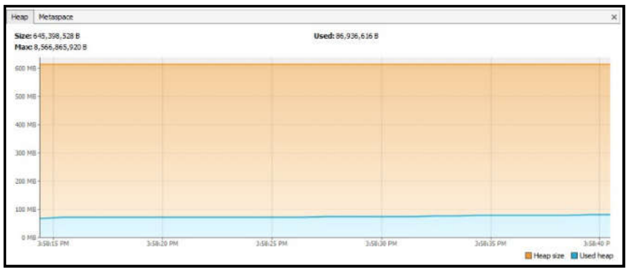

Figure 6 OAuth2.0 Memory profile

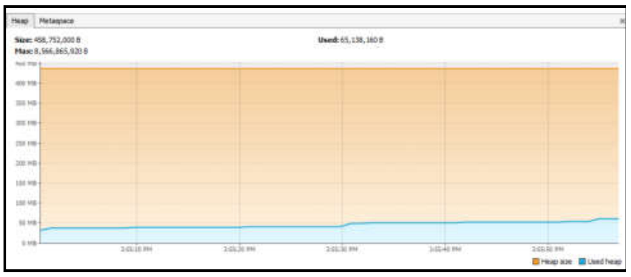

Figure 7 OpenID Connect Memory Profile

The above two figures shows that the OAuth2.0 profile uses more memory (64 MB) as compared to OpenID Connect (45 MB). This is because it requires to refresh the tokens at intervals and also the bearer tokens are stored which are used for authorization mechanism. Though it may consume memory of the system however this might be considered as beneficial due to the fact that it complicates the method of guessing the tokens from excessive memory. Thus avoiding Brute-force attacks. OpenID Connect on the other hand has no such requirement and thus consumes relatively less memory. Several of the user modifiable fields in OAuth are fixed in OpenID Connect and hence the implementation complexity is thus reduced. But this memory consumption profile makes an important component of the microservices. Attempts should be made to make it as light weight as possible, due to the limited resources available with them. Also care must be taken to make the server store as less as possible data about the clients.

The comparison thus formulated are as follows:

Table 2 Comparison of implemented protocols

Feature	OAuth 2.0	OpenID Connect
Service Provider & Identity Provider Initiated Login	Yes	Yes
High security Identity tokens	No	Yes
Users' consent before sharing attributes	Yes	Yes
Token contains user identity information	No	Yes
Distributed & Aggregated Privileges	No	Yes
Dynamic Outlines (Client discovery & on-boarding)	No	Yes
Session timeout	No	Yes

CONCLUSION

Microservices are small, autonomous services concentrated on a single task, hence it is very essential to unburden them of concerns related to authentication and authorization. OpenID Connect caters as the best protocol for both authentication along with authorization via OAuth 2.0. This enables the services to function without the overhead of maintaining databases for storing usernames and passwords. Furthermore it also provides the best in market security combatting several serious threats to privacy and integrity of the end users. This paper implemented the two models and provided a comparison and evidences supporting the use of OpenID connect for authentication in microservices.

References

1. Alshuqayran, N., Ali, N., & Evans, R. (2016, November). A Systematic Mapping Study in Microservice Architecture. In *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on* (pp. 44-51). IEEE.
2. Blazquez, A., Tsiatsis, V., & Vandikas, K. (2015, May). Performance evaluation of openid connect for an iot information marketplace. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st* (pp. 1-6). IEEE.
3. Domenech, M. C., Comunello, E., & Wangham, M. S. (2014, October). Identity management in e-Health: A case study of web of things application using OpenID connect. In *e-Health Networking, Applications and Services (Healthcom), 2014 IEEE 16th International Conference on* (pp. 219-224). IEEE.
4. Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.
5. Hassan, S., & Bahsoon, R. (2016, June). Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap. In *Services Computing (SCC), 2016 IEEE International Conference on* (pp. 813-818). IEEE.
6. "How We Ended up with Microservices." How We Ended up with Microservices. N.p., n.d. Web. 04 Jan. 2017.
7. Jaramillo, D., Nguyen, D. V., & Smart, R. (2016, March). Leveraging microservices architecture by using Docker technology. In *SoutheastCon, 2016* (pp. 1-5). IEEE.
8. Matt Stine. www.mattstine.com/microservices/ N.p., n.d. Web. 02 Jan. 2017
9. Microservices: Decomposing Applications for Deployability and Scalability." InfoQ. N.p., n.d. Web. 02 Jan. 2017.
10. "Microservices." Martinfowler.com. N.p., n.d. Web. 04 Jan. 2017.
11. "Microservices: Five Architectural Constraints." Nirmata. N.p., n.d. Web. 02 Jan. 2017.
12. Naik, N. (2016, October). Connecting google cloud system with organizational systems for effortless data analysis by anyone, anytime, anywhere. In *Systems Engineering (ISSE), 2016 IEEE International Symposium on* (pp. 1-6). IEEE.
13. Newman, S. (2015). *Building microservices*. " O'Reilly Media, Inc.".
14. "The Netflix Tech Blog." The Netflix Tech Blog. N.p., n.d. Web. 04 Jan. 2017.
15. Werner, J., & Westphall, C. M. (2016, June). A model for identity management with privacy in the cloud. In *Computers and Communication (ISCC), 2016 IEEE Symposium on* (pp. 463-468). IEEE.
16. www.mattstine.com/2014/06/30/microservices-are-solid/ Matt Stine. N.p., n.d. Web. 02 Jan. 2017.

How to cite this article:

Shagufta N. Shaikh and Sunil B. Mane (2017) ' Authentic Techniques Of Authentication In Microservices', *International Journal of Current Advanced Research,* 06(04), pp. 3342-3345.
DOI: http://dx.doi.org/10.24327/ijcar.2017.3345.0267