Securing RESTful Web Services using Multiple JSON Web Tokens

Pedro Mestre, *Member, IAENG,* Rui Madureira, Pedro Melo-Pinto, and Carlos Serodio, *Member, IAENG*

***Abstract*—Because of their stateless property RESTful web services cannot use session based authentication, therefore other authentication and authorization techniques must be used instead. Basic HTTP Authentication Scheme or HTTP Digest Access Authentication are two valid options, however these techniques are not very suitable when performance is a key issue, because the user credentials must be checked for every HTTP transaction (e.g. using a database). An alternative method is to use token based authorization, for example using JSON Web Tokens. The client credentials are verified once by an authentication service, which issues a token that the client uses to access the services. While the token is valid there is no need to check the user credentials again. In this paper it is presented a system based on multiple JSON Web Tokens, one per transaction to prevent replay attacks, which supports anytime token revoking, based on distributed token issuing and validation. The proposed token based system, when compared with those that do not use tokens had a better performance. When a single service provider is used, in our test conditions, it is 198% faster than authenticating all requests using a database.**

***Index Terms*—RESTful, web services, authentication, authorization, token, multiple tokens, JSON Web Token.**

I. Introduction

ACCORDING to [1] RESTful web services, which are lightweight web services, are particularly well suited for creating APIs for clients spread out across the Internet and some authors have been using RESTful web services in projects related to agriculture and farm management [2],[3]. These were therefore chosen by the authors of the preset paper to create remote data access services in projects also related to agriculture and wine production.

In these projects a set of distributed applications and sensors are used to collect georeferenced data from vineyards, such as multispectral images of grape bunches, photographies of vine leaves, micro-meteorological data, textual information inserted by staff (e.g. anomalies reporting), etc.

These data are collected using wireless sensors, embedded computers and mobile devices, and uploaded to a server for storage and processing. To the above mentioned data it can also be added additional information, for example by the vineyard manager, such as comments and vineyard works schedule. Besides data upload these web services also support data access anytime and anywhere.

In those projects web services and applications are being developed to work as generic as possible, to allow its use in several vineyards (at the moment) and crop types (in the future), and used by several business clients/partners. The objective is to create a cloud service for agricultural services, in a first stage for the Douro Region in the North of Portugal, that can serve both the Scientists and agriculture-related Businesses. Because data belongs to several business clients a major concern in this project is to ensure that access to web services is made only by authorized principals (people and devices).

Because one of RESTful web services constraints is that communications between the client and the server must be stateless [4], no session information can be stored in the server, and therefore session based authentication methods cannot be used.

If sessions are not used, then after every transaction all relevant data related to the session (and that will be needed later) must be sent to the client. This stateless nature of RESTful web services plays an important reole in service scalability.

Because servers do not store session data this means that whenever a client requests a service the server will not ”remember” the client. The server must then be able to authenticate the client and check its authenticity for every HTTP transaction.

Client authentication can be made by sending authentication information in every HTTP request using for example the Basic HTTP Authentication Scheme, as described in RFC7617 [5], or the HTTP Digest Access Authentication, as in RFC7616 [6].

In both cases the server must have access to the user credentials, stored for example in a relational Database (e.g. MySQL, PostgreSQL, etc.) or using LDAP (Lightweight Directory Access Protocol) [7], just to mention two examples.

This is a very simple method, easy to implement, but that has scalability issues. For example let us consider the authentication based on a relational Database: its performance will depended on the database technology and on the size of the tables [8]. If every transactions needs that the user authentication information is checked or retrieved from the database, if the number of clients and transactions increase, the service performance will start to decrease.

Manuscript received March 1, 2017; revised 3 April, 2017.This work was supported by the project ”VitiNov” - PA 52306, funded by he Agricultural and Rural Development Fund (EAFRD) and the Portuguese Government by Measure 4.1– Cooperation for Innovation PRODER program – Rural Development Programme.

P. Mestre is with Centre for the Research and Technology of Agro-Environmental and Biological Sciences, CITAB, University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, and, Algoritmi Research Centre, Guimarães, Portugal (phone: +351-259350363; email: pmestre@utad.pt)

R. Madureiria is with University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, email: rccmadureira@gmail.com)

P. Melo-Pinto is with Centre for the Research and Technology of Agro-Environmental and Biological Sciences, CITAB, University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, and Algoritmi Research Centre, Guimarães, Portugal (email: pmelo@utad.pt)

C. Serodio is with Centre for the Research and Technology of Agro-Environmental and Biological Sciences, CITAB, University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, and Algoritmi Research Centre, Guimarães, Portugal (email: cserodio@utad.pt)

ISBN: 978-988-14047-4-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Other authentication methods that can be used by services to authenticate clients include the use of tokens: the user requests authentication to an authentication service; the authentication service sends a token to the client; the client uses this token to access the service. Credentials are sent only at the beginning or after the token has expired, not in every transaction.

To implement such an authorization system to validate client requests to web services we have many options, including the use of simple tokens such as JSON Web Tokens (JWT) [9] or use a more complex infrastructure such as OAuth 2.0 Authorization Framework [10].

The objective of this work is to develop a scalable authentication and authorization system based on tokens that allows anytime token revocation, that complies with the stateless constraint of RESTful web services and that prevents replay attacks. Because of its simplicity, instead of adapting an existing framework to the needs, authors decided to build a custom authorization system based on JWT.

The solution presented in this paper is based on multiple tokens, one per transaction, to avoid replay attacks (without the need of session tokens) and that can be easily revoked without the need of a centralized token management system. From the clients' point of view, this is a lightweight system that can be easily integrated in IoT (Internet of Things) devices, Web and mobile applications, and M2M (Machine-to-Machine) systems.

II. Authorizing Access to Web Services using JSON Web Tokens

JWT is an open standard, RFC7519 [9], which defines a way of sending signed information, as a JSON object, between parties. Advantages of using JWT includes: it is compact in comparison to XML based solutions such as SAML (Security Assertion Markup Language) [11]; information can be verified and trusted because tokens can be digitally signed.

A JSON Web Token is made of three parts:

- Header – which usually has two fields: the type of the token (JWT) and the hashing algorithm;
- Payload – that contains the token reserved, public and private claims.
- Signature – used to verify if the token can be trusted or not.

A typical header for a token signed using HMAC SHA256 would be:

```
{
"typ": "JWT",
"alg": "HS256"
}
```

Statements about the client (claims) are sent in the payload. These can be:

- Reserved, which are predefined recommended claims: Expiration Time (exp), Not Before (nbf), Issued At (iat), JWT ID (jti), Issuer (iss), Audience (aud), and Subject (sub);
- Public claims that can be freely defined;
- Private claims, which are not part of Reserved or Public claims, and that a producer and consumer of a JWT may agree to use.

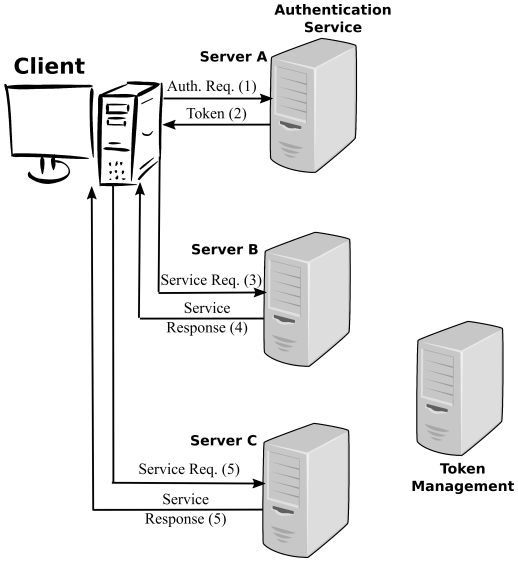

Fig. 1. Simplified Diagram of Authentication and Service Access using a Single Token.

An example of a Payload using some of the Reserved Claims would be:

```
{
"nbf": 1499238000,
"exp": 1499450400
}
```

The above listing is a very simple example that uses only two claims (Not Before and Expiration Time). This token is valid from 05 Jul 2017 07:00:00 GMT until 07 Jul 2017 18:00:00 GMT, i.e., during WCE 2017.

Signature of a JWT token can be obtained either by using HMAC algorithm (as above) or a public/private key pair using RSA.

The resulting token is the concatenation of the above three parts (Header, Payload and Signature), encoded in base64 and separated by dots. As an example the above presented example token, signed using a shared key "WCE2017" would look like the example below:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJu
YmYiOjE0OTkyMzgwMDAsImV4cCI6MTQ4ODI4ODYxN
SwianRpIjoiZDA2NzQyMjYtNzZhZC00MTAzLTg3ZD
MtZWZkYzJlNWYwNjE3IiwiaWF0IjoxNDg4Mjg1MDE
1fQ.e-kIrSZgZ4hPVX4psgHGqNMPhPzB-WcLxx2V0
RbFxvk
```

Please notice that line breaks were added for better visualization.

A. Using a Single Token

A simplified diagram exemplifying how JSON Web Tokens can be used for authentication and authorization is depicted in Fig. 1. In this example we assume that all servers are trusted and can verify the token signature:

- (1) – First, the client makes an Authentication Request to the Authentication Service, sending its credentials to the server. This request can be made using any non-token based method (e.g. Basic Authentication). While

ISBN: 978-988-14047-4-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the token is valid there is no need to repeat this step again;

- (2) – Credentials are checked, for example using a Database, and if the client can be trusted a token is issued and returned to the client. This token can contain claims about the client permissions (e.g. which services the client can access), an unique token ID and token expiry time;
- (3, 5) – As long as the token is valid, to access services (without the need of re-authentication) the client only needs to "present" it to the service providers;
- (4, 6) – If the server can verify the token signature, and the client has the correct permissions for the service it is requesting, the server executes the service and sends the response to the client. If the server cannot verify the token signature, or if the token has already expired, or the client does not have the correct permissions, the server response will be a "403 Forbidden" HTTP response.

In the above example, because both servers can verify the token signature, the client can access services of any server using a single token, provided that it has the right permissions.

The client only has to send authentication information once, and there was also only a single access to the users database. All other transactions are made without the need to access to user information. It is then expected that the performance of this method is better than those that rely on user authentication for every transaction.

Obviously that the diagram presented in Fig. 1 is missing a key player for the security of the system: what happens when we need to revoke a token? We need a token Management entity that stores information about the issued tokens, as presented in Fig. 2.

When a service is requested by the client, the server must ask to this Token Management entity if the token is valid. If a device is lost or compromised, the token can be revoked, and the server will be informed by Token Management that the token is not valid anymore.

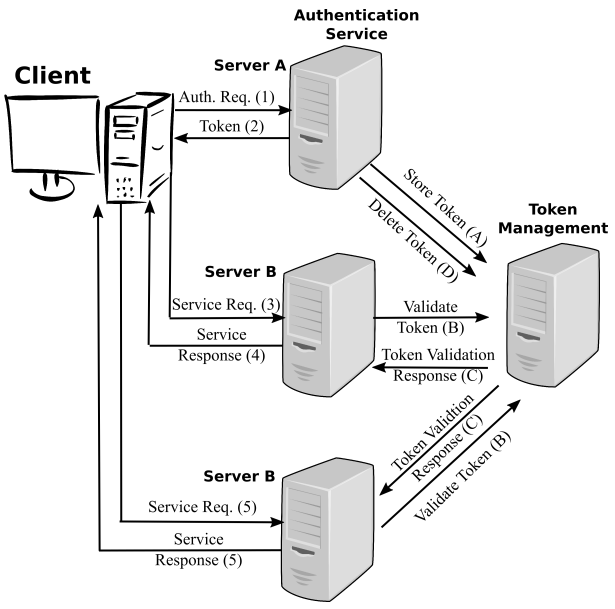

Fig. 2. Authentication and Service Access using a Single Token.

This implies that extra steps are needed (Fig. 2): (A) When the token is issued it must be sent to the Token Management; (B) When a client requests a service, the server must check the token validity; (C) The Token Manager must inform the server if the token is still valid; (D) If the token is revoked, the Token Manager must remove it from its valid tokens list.

B. Using Multiple Tokens, one per Transaction

Relying on the Token Management service to validate the token would create a single point of failure, and for every transaction this server would be contacted by the service providers. An increase on the number of clients and/or transactions would also increase the number of token validation requests. This means that a cluster of Token Management server would be needed to cope with performance and service availability issues.

Grouping transactions or define a time limit between token validation verifications could solve the issue related to the number of requests. However if a token is revoked there will be a propagation delay.

So the solution that authors propose in this paper is that token issuing and validation is distributed, i.e., each server can issue a new token and validate its own tokens. When a server receives a client request, if the token did not expire yet, the server first verifies who issued that token. The server will then contact the token issuer and checks if it is still valid. A token is valid if it was not revoked or used in a previous transaction.

URL of the token validation service is one of the public claims inserted in the token by its issuer. Servers can only send token validation requests to trusted servers.

This is not the same as Refresh Tokens, because Refresh Tokens are used to obtain a new access token when the current access token becomes invalid or expires [10]. In this case the a new token is issued for every transaction. Because there is a token per transaction, replay attacks can be prevented.

The working principle of the proposed system is exemplified in Fig. 3:

- (1) – The client sends its credentials to the Authentication service;
- (2) – Credentials sent by the client are verified (e.g. using a Database) and if the principal is positively authenticated then a new token is issued. This token is stored locally in a token cache for later validation;
- (3) – The token is sent to the client;
- (4) – When the client makes a request to a server (in this example Server B) it has to send the previously received token. If the server can verify the token signature and it has not expired already it will then check which trusted server has issued the token (in this example it was Server A).
- (5) – Server B will contact the token issuer (Server A) requesting the token validation;
- (6) – If server A finds the token in its token cache (the token is valid) then the token is deleted to prevent further attempts from clients to use the same token;
- (7) – Server B is informed if the token was found in the tokens cache, i.e., if the token is still valid;
- (8) – The server will issue a new token and adds it to its token cache (for future validation);

ISBN: 978-988-14047-4-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

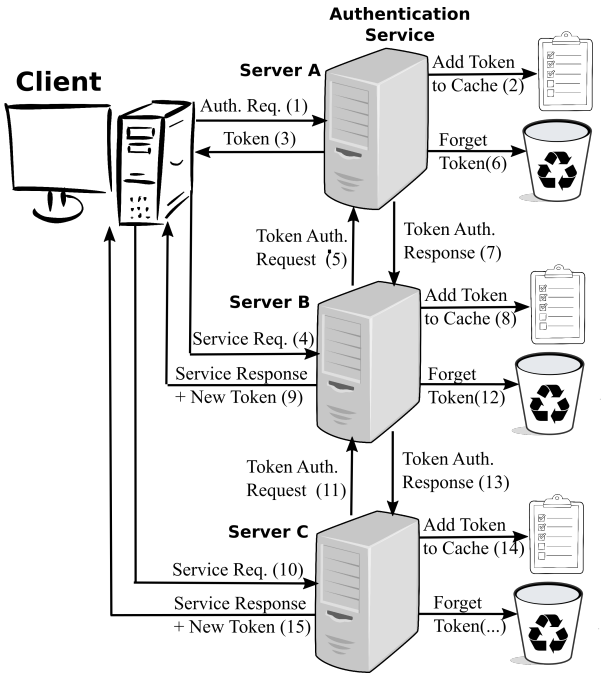

Fig. 3. Authentication and Service Access using Multiple Tokens.

- (9) – Service requested by the client is executed and when the server sends the response to the client it also sends the new token that the client must use in its next service request.

If the next client request is made to the same server (step 4 again), then the server will check for the token on its own cache (steps 5 and 6 are not executed) and deletes the token form its cache (8). If instead the request is made to another server (e.g. Server C), then the above process is repeated: (10) Request is sent to server; (11) Server asks to the issuer server for the token validity; (12) token is deleted form the cache; (13) Token validity response is sent back to the requesting server; (14) A new token is issued and stored in the local cache; (15) The new token is sent to the client together with the service result.

Besides information regarding to the principal permissions tokens must also have an expiry time. Whenever a token expires it must be removed from the cache. This can be achieved using a service that checks the expire time of every token stored in the token cache. Even in heavy load conditions this is a very fast operation. For example, if an Hash Table is used to implement the cache, search for items to remove is an operation that has a cost of O(n) and removing an item as a cost of O(1). This means that in the worse case scenario, when all tokens have expired, this ”cleanup” operation will take O(n).

At anytime if there is a security problem and a token must be revoked, it is only needed to send a message to the servers requesting them to ”forget” any token of that client. This means that an unique identifier for the client must be sent as a claim in the token.

III. Tests and Numerical Results

In this section are presented the numerical results obtained with performance tests made to the proposed system. These tests were made using a client running multiple threads and each thread making several service requests.

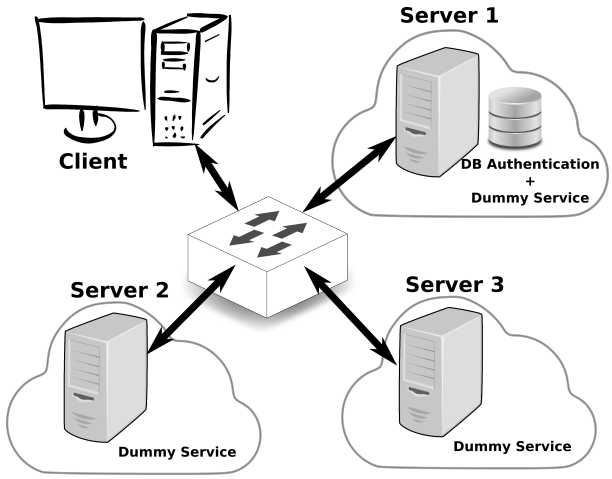

Fig. 4. Testing Scenario

A. Testing Scenario

To implement the proposed authentication service any server technology could be used. Because this authentication and authorization system will be used in projects that are already using Java, the services were implemented Java Enterprise Edition 7.

In Fig. 4 it is presented the logical diagram of the testing scenario, which was implemented using virtual machines (each server or client is a virtual machine). All those virtual machines are running in the same server, and to each virtual machine it was allocated 4Gbytes of RAM and a processor core (Intel(R) Xeon(R) E5-2620@2.00GHz). These resources were not shared, i.e., the test server has enough RAM and CPU cores for all virtual machines.

Client and servers are connected using a virtual switch. To verify how the connection speed between the peers would influence the results, for each set of tests the switch speed was set to: no limit; 100Mbps; 1Gbps. The last two to simulate when services and clients are physically implemented in different computers.

In this scenario all servers and clients are running Ubuntu Linux Server 16.10, 64 bit version, web services are running on Glassfish 4.1.1 Opensource Edition using Java 8 (1.8.0_121). To create and verify the tokens it was used JJWT (JSON Web Tokens for Java and Android).

Server 1 is running the Authentication Service and a ”dummy” service. The Authentication Service is used by the client to get the token that will enable access to services provided by Server 2 and Server 3, and to validate the tokens that it has issued (for multiple tokens). The ”dummy” service on Server 1 requires user credentials to be sent in every transaction, and it checks those credentials in a MySQL database table which has 10.000 user records.

Both Server 2 and Server 3 have the following services: a dummy service that uses a single token; a dummy service that used multiple tokens (and issues a new token when the service is called); a token validation service.

B. Database Authentication Tests

This first set of tests consisted in sending 250 requests per client to a web service that authenticates the user credentials using a database. Results of these tests are used as reference

ISBN: 978-988-14047-4-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

values to assess the performance improvement, that is expected, when token-based solutions are used. In Table I are presented the time per transaction values, in ms, that were obtained.

TABLE I
DATABASE AUTHENTICATION.

	Number of Clients					
Conn. Speed	10	20	30	40	50	100
No Limit	1,409	1,406	1,400	1,394	1,395	1,384
100Mbps	1,415	1,405	1,402	1,397	1,395	1,389
1Gbps	1,407	1,404	1,399	1,390	1,387	1,383

C. Using a Single Token

The second set of tests consisted in authenticating the client using the Authentication Service and then use a single token to access the web services. Results presented in Table II (in μs) were obtained using a single service provider to which the client made 1000 requests.

TABLE II
SINGLE TOKEN, SINGLE SERVER.

	Number of Clients					
Conn. Speed	10	20	30	40	50	100
No Limit	446	442	446	443	448	452
100Mbps	449	443	446	444	446	443
1Gbps	451	444	452	440	450	445

Similarly to the above tests, values in Table III were obtained using a single token, however in this case service requests were sent to two servers. All clients made 500 requests to each server. Client requests were sequential, and the client only sent a new request after receiving the response to the previous request. Values in Table III are in μs.

TABLE III
SINGLE TOKEN, TWO SERVERS.

	Number of Clients					
Conn. Speed	10	20	30	40	50	100
No Limit	264	266	256	258	253	250
100Mbps	267	260	256	255	253	249
1Gbps	274	262	256	254	250	250

D. Using Multiple Tokens

This third set of tests consisted in using multiple tokens. The Client sent its credentials to the Authentication Service that issued the first token, and then each service provider sent a new token whenever the client requests a service.

Table IV presents the time per transaction when services are requested to a single server (1000 requests) and Table V presents the values obtained when two servers were used. Is the latter case clients made 500 requests to each server. In both cases clients only sent a request after receiving a response to the previous request. Values in the tables are in μs.

TABLE IV
MULTIPLE TOKENS, ONE SERVERS.

	Number of Clients					
Conn. Speed	10	20	30	40	50	100
No Limit	476	470	473	462	466	464
100Mbps	481	470	468	463	468	467
1Gbps	481	473	476	466	461	466

TABLE V
MULTIPLE TOKENS, ONE SERVERS.

	Number of Clients					
Conn. Speed	10	20	30	40	50	100
No Limit	563	589	585	585	591	590
100Mbps	563	585	583	583	592	587
1Gbps	560	581	584	579	593	588

IV. DISCUSSION

As it was already expected worse performance was obtained using authentication of every request, based on a Database, which had an average time per transaction of 1.398ms. As it can be seen in the plot of Fig. 5, when a single token (and single server) the transaction response time decreases to approximately one third ($446\mu s$). If two servers are used the mean time per transaction is reduced even further (to $257\mu s$) as a consequence of load balancing between servers. The actual time per transaction does not decrease, however each server has half of the requests therefore the mean value is lower.

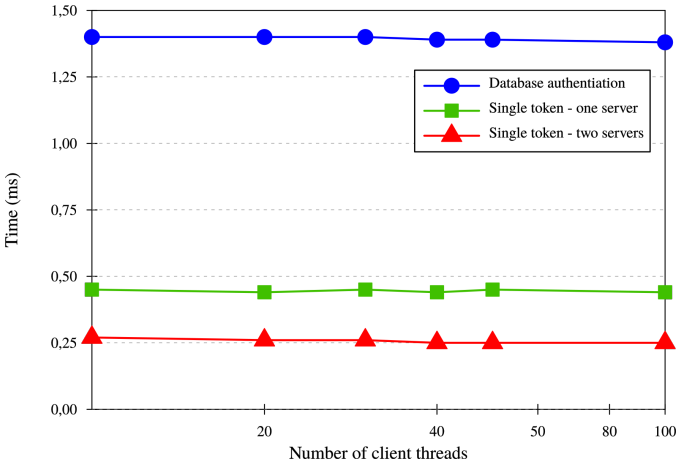

Fig. 5. Performance Comparison of Database Authentication vs Single Token.

ISBN: 978-988-14047-4-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

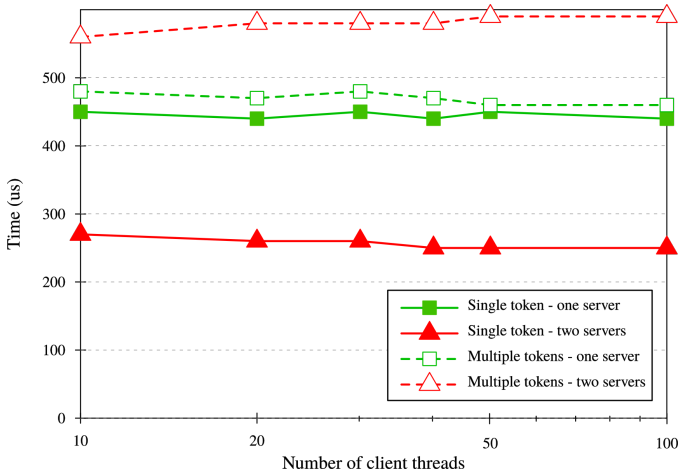

Fig. 6. Performance Comparison of Using a Single Token vs Using Multiple Tokens.

If we compare the performance of ”Single Token” vs ”Multiple Tokens” using only one server (Fig. 6) we can see that we have similar performance. There is only a 5% between the obtained values. This can be easily explained because there is only a single server, and the slight time increase is because of the time needed to issue the new token. However when two servers are used, as expected, the performance will be worse. We now have the network between service provider and token validation service. If we compare the time per transaction with that obtained using a ”Single Token” and one server there is an increase of 32% and an increase of 126% in comparison to ”Single Token” with two servers.

V. Conclusion and Future Work

In this paper it was presented a token based web services authentication and authorization system using multiple JSON Web Tokens. Although the proposed system has a worse performance than those that use a single token, it has the advantage of enabling fast token revoking if a device is compromised. Also Token Management replication is an intrinsic characteristic.

On the other hand, if we compare the performance of ”Multiple Tokens” with that of Database authentication we can conclude that it has a much better performance. Time per transaction of the latter is worse by 140%. If we compare the results using a single server, that value rises to 198%.

To be noticed that the proposed system was not negatively affected by the available bandwidth. In fact for all tests it has no or little influence on the performance, despite the increase of the number of messages exchanged between the servers.

Because symmetric key algorithms are faster than those that use asymmetric keys [12], authors have used the first option to create the token signatures. However this raises the question of how to distribute those shared symmetric keys with ”in production” trusted servers.

As future work authors plan to implement a key sharing feature, that can be based on the solution presented by Foltz and Simpson in [12] to share encryption keys for documents.

Such a key sharing system can also be extended to other features that need to be implemented such as the distribution the trusted servers list.

Another feature that could be very useful, and that would increase the performance of token revoking is to implement a multicast message exchange between the servers, that are in the same data-center.

Acknowledgment

References

[1] E. Jendrock, R. Cervera-Navarro, I. Evans, K. Haase, and W. Markito. The Java EE 7 Tutorial: Volume 2. Oracle. [Online]. Available: https://docs.oracle.com/javaee/7/tutorial/

[2] M. Arroqui, C. Mateos, C. Machado, and A. Zunino, “{RESTful} web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones,” *Computers and Electronics in Agriculture*, vol. 87, pp. 14 – 18, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169912001305

[3] A. Kaloxylos, A. Groumas, V. Sarris, L. Katsikas, P. Magdalinos, E. Antoniou, Z. Politopoulou, S. Wolfert, C. Brewster, R. Eigenmann, and C. M. Terol, “A cloud-based farm management system: Architecture and implementation,” *Computers and Electronics in Agriculture*, vol. 100, pp. 168 – 179, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169913002846

[4] I. Porres and I. Rauf, “Modeling Behavioral RESTful Web Service Interfaces in UML,” in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC ’11. New York, NY, USA: ACM, 2011, pp. 1598–1605. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982521

[5] J. Reschke, “The ’Basic’ HTTP Authentication Scheme,” Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7617, September 2015.

[6] R. Shekh-Yusef, D. Ahrens, and S. Bremer, “HTTP Digest Access Authentication,” Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7616, September 2015.

[7] J. Sermersheim, “Lightweight Directory Access Protocol (LDAP): The Protocol,” Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 4511, June 2006, http://www.ietf.org/rfc/rfc4511.txt. [Online]. Available: http://www.ietf.org/rfc/rfc4511.txt

[8] A. Saikia, S. Joy, D. Dolma, and R. Mary, “Comparative Performance Analysis of MySQL and SQL Server Relational Database Management Systems in Windows Environment,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 3, 2015.

[9] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7519, May 2015, http://www.ietf.org/rfc/rfc7519.txt. [Online]. Available: http://www.ietf.org/rfc/rfc7519.txt

[10] D. Hardt, “The OAuth 2.0 Authorization Framework,” Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 6749, October 2012, http://www.ietf.org/rfc/rfc6749.txt. [Online]. Available: http://www.ietf.org/rfc/rfc6749.txt

[11] B. Campbell, C. Mortimore, and M. Jones, “Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants,” Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7522, May 2015.

[12] K. E. Foltz and W. R. Simpson, “Simplified Key Management for Digital Access Control of Information Objects,” in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2016, WCE 2016, 29 June - 1 July, 2016, London, U.K.*, 2016, pp. 413–418.

ISBN: 978-988-14047-4-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)