A Novel JWT Revocation Algorithm

László Viktor Jánoky, János Levendevoszky, Péter Ekler

Abstract: JSON Web Tokens (JWT)[1] provide a scalable, distributed way of user access control for modern web-based systems. The main advantage of the scheme, is that the tokens are valid by themselves - through the use of digital signing - also imply its greatest weakness. Once issued, there is no trivial way to revoke a JWT token. In our work, we present a novel approach for this revocation problem, overcoming some of the problems of currently used solutions.

Keywords: JWT, JSON Web Tokens, User access control

1 Introduction

In our previous paper in the field, titled as An analysis on the revoking mechanisms for JSON Web Tokens[2], we examined the currently used solutions of token revocation and laid out the mathematical foundations and introdued a novel approach. In this paper, we further elaborate on the new solution and provide general guidelines how to use it in a real-world application.

This paper is structured as follows: (I) in this first section, we quickly recap the currently used revocation schemes and their main characteristics, (II) in the second section, in which we provide a detailed description of our new approach and give some recommendations for its real-world use, (III) the third section deals with the evaluation of performance in different cases, comparing our solution with existing one, and finally (IV) the fourth and final section wraps the discussion by providing an overview of the work done.

A JWT token used to determine access for a protected resource is called an access token. The token is usually digitally signed, or otherwise cryptographically secured [3], in both cases we simply refer to the signing key or the public key as secret. In most scenarios, the access tokens are issued along with a second, more traditional; server-side token called a refresh token. This second token makes it possible for the client to acquire a new access token in the future.

When a client logs out from the system, the refresh token is destroyed, and existing JWT tokens are revoked. There are three main methods of this revocation that are currently used[4].

Short-lived tokens: Each generated JWT token has a finite, usually very short lifespan. In this scheme, a token is never directly revoked, but the means of acquiring new tokens are made unavailable, hence when the short lifespan runs out, no further access is possible to the system.

Blacklist: In case of a blacklist, revoked tokens are placed in a shared location (typically a database), where each consuming service can check for invalidated tokens. The big downside of this approach is that it requires data access for each request served - even for ones with valid tokens thus, the validity of the token can no longer be determined in itself.

Secret change: A rarely used solution for invalidation, is the changing of the cryptographic secret used to issue and check the validity of tokens. Changing this secret leads to all tokens being revoked, but still logged in users can apply for new ones using their refresh token.

2 The revocation algorithm

Our novel revocation strategy is based on the extension of the third option, which is changing the secret. In this section, we give a quick overview of this approach.

2.1 Basic principle

When a JWT secret is changed, all the tokens issued with it become invalid. In practice, this means that if a user logs out, every other user in the system must request a new token.

This effect can be controlled by arranging the users in groups (for example, by hashing their usernames) and assigning a different secret for each group. If a token is revoked in a group, only tokens signed with the group secret will be revoked, instead of all the tokens.

As log-outs are typically infrequent events, one can use statistical methods (such as described in our previous paper), to calculate an optional group size, which minimizes the number of unnecessary revocations, while maintaining a manageable number of secrets.

2.2 Propagating secret change events

With this method, revocation is instantaneous, and the basic premise of JWT tokens remain intact, that the tokens are valid by themselves.

This solution requires the existence of a channel for propagating the information about the change of the JWT secret. The channel must be available between the token issuer and each service consuming the tokens.

For cases when such a channel is unavailable, we have come up with an alternate solution. In this approach the JWT Secret is generated as a rolling code by a pseudo random number generator[6], each service keeping track of the currently active value. When a token is revoked and the group's secret is changed, the new tokens are issued with the new secret. When a service, still using the old code, receives a token signed with the new secret (a next value from the rolling code), it will also update the secret accordingly.

This method provides eventual consistency for the system in the long run, without the need for a dedicated channel for JWT secret change event prorogation. As a trade-off, the instantaneous revocation is lost, a token is only revoked after the code is rolled (another token, using the new secret is received).

2.3 Cost model of running the algorithm

In our previous work on the topic, we prove that any system can be parametrized in a way, for our solution to be better in terms of performance than the previous solutions.

In this paper, we further examine the performance and the costs of our solution and provide a mathematical model to aid in system design and capacity planning.

To accurately model the performance impact of different solutions, a common framework must be set. As the basis of this framework, we determined a set of basic operations, which make up each approach. The costs of these operations can be parametrized, which can be based on estimations or measurements. The following main costs are identified:

- C_i **(Issue cost):** the cost of issuing a new token.
- C_v **(Validation cost):** the cost of checking the validity of a token.
- C_c **(Communication cost):** the cost of system modules communicating with each other.
- C_d **(Data access cost):** the cost of accessing data stored in a persistent storage.

In order to predict the performance of a system, it is not enough to know the cost of these atomic operations, one must also calculate how many times they will occur. The performance cost of a system comes from the clients consuming its service. By characterizing the client sessions, one can come up with predictions for their impact on the system[5]. In order to calculate the former, the following properties must be known (by measurements or estimation).

- N**:** the number of clients in the system.
- $f_i(t)$**:** probability distribution of client session lengths.

- r**:** average number of protected resource access / client / second.

As we have demonstrated in our earlier work on the topic, from these metrics, one can calculate the average time between token revocations in a group of clients. This value is denoted as T_{rvk}.

Knowing both the cost and occurrences of typical operations in the system, one can come up with a cost function, which describes the average cost of operation.

3 Performance evaluation

To predict the performance characteristics of a system, one must first determine the costs associated with the operations defined in the previous section. The second step is to measure the characteristics of the client population. The third step is to choose a revocation algorithm.

These three steps together determine the overall performance metrics. Each revocation algorithm has a unique cost function, which maps the client pool behavior to system operations, which in turn are used to calculate the overall performance cost of a given solution. Some revocation algorithms have variable parameters, which can be optimized trough the usual means.

3.1 Short-lived tokens

The short-lived approach has one parameter, T_{life}, which denotes the lifetime of a token. As for maximizing the performance of this approach, this T_{life} should be chosen as the longest tolerable time for token revocation after logout.

The overall cost function consists of two parts, the cost of validating the tokens of the incoming requests, and the cost of issuing new tokens to replace the expiring ones.

$$C = (N * r * C_v) + (N * \frac{1}{T_{life}} * C_i)$$

3.2 Blacklist

In the case of a blacklist, the main cost factor comes from the necessity to check whether a token is on the blacklist for each request. This extra checking makes this approach the worst in terms of scaling in the case of an increasing number of requests.

After accounting for this factor, there are no other costs associated with this method. Therefore the cost function can be defined as the following.

$$C = N * r * C_v * C_d$$

3.3 Secret change

In case of a secret change, the baseload of authorizing incoming request is still present, but it is accompanied by the load of new token generation for each client in case of each revocation.

$$C = (N * r * C_v) + (N * \frac{1}{T_{rvk}} * C_i)$$

Notice that the formula is very similar to the short-lived cost function. This is not a coincidence; in both cases, the number of token revocations depends heavily on the average lifespan of a token. In the first case its purely determined by the age of token itself, while in the second case, client logout events trigger it.

3.4 The cost evaluation of our method

As our method builds on the secret change event, the cost function is similar too. The main difference being, the introduction of parameter K, which denotes the number of groups the clients are separated to. Because of this separation, the T_{rvk} calculated for the whole client population size must be recalculated to the number of $\frac{N}{K}$ clients. This value is denoted by T_{rvk}. As K increases, T_{rvk} monotonously increasingly approaches the mean value of $f_i(t)$.

$$C = (N * r * C_v) + (K * \frac{1}{T'_{rvk}})(\frac{N}{K} * C_i + C_c)$$

4 Overview

In this paper, we described the main approaches of JWT revocation and introduced our novel solution. We provided a toolset for characterizing different systems based on the cost of common operations when dealing with JWT tokens. We outlined the necessary parameters to measure in a client population of such a system. We determined the cost functions for each solution, based on the previously described characteristics and client behaviour.

With the mathematical framework at hand, one can find the optimal revocation solution for any system, by choosing the minimal cost function. In cases where the function has additional variable parameters, traditional approaches like linear search can be used to find the optimal solutions.

Ultimately, we hope that our work will aid capacity planning and system design of distributed systems, as the JWT based solutions have the highest potential in this area.

Acknowledgements

This work was supported by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC) and by the János Bolyai Research Fellowship of the Hungarian Academy of Sciences.

References

[1] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC Editor, RFC 7519, May 2015.

[2] L. V. Jánoky, J. Levendovszky, and P. Ekler, "An analysis on the revoking mechanisms for JSON Web Tokens," International Journal of Distributed Sensor Networks, vol. 14, no. 9, p. 1550147718801535, Sep. 2018, doi: 10.1177/1550147718801535.

[3] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," RFC Editor, RFC 7515, May 2015.

[4] dWTV, "Learn how to revoke JSON Web Tokens," developerWorks TV, 27-Jul-2017. [Online]. Available: https://developer.ibm.com/tv/learn-how-to-revoke-json-web-tokens/.

[5] M. Arlitt, "Characterizing web user sessions," ACM SIGMETRICS Performance Evaluation Review, vol. 28, no. 2, pp. 50â63, 2000.

[6] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," SIAM journal on Computing, vol. 13, no. 4, pp. 850â864, 1984.