Acta Cybernetica **25** (2021) 307–318.

Evaluating the Performance of a Novel JWT Revocation Strategy[*]

László Viktor Jánoky[ab], Péter Ekler[ac], and János Levendovszky[de]

Abstract

JSON Web Tokens (JWT) 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. To compare our solution to the established solutions, we also introduce the mathematical framework of comparison, which we ultimately test using real-world measurements.

Keywords: JWT, JSON Web Tokens, user access control

1 Introduction

In the field of web application security, JSON Web Tokens play an increasingly significant role. They are very well suited for application in distributed systems, as their validation can be done by the consuming service, without the need for central access to a trusted source. This property, however, also means that there is no easy way to revoke a token once it has been issued.

This paper overviews the current revocation strategies and introduces a mathematical framework for comparison to help system designers find the optimal solution. The mathematical framework is then validated with measurements on a real

[*] The research reported in this paper was supported by the BME Artificial Intelligence TKP2020 IE grant of NKFIH Hungary (BME IE-MI-SC TKP2020). Project no. FIEK_16-1-2016-0007 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Centre for Higher Education and Industrial Cooperation – Research infrastructure development (FIEK_16) funding scheme.

[a] Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary

[b] E-mail: janoky.laszlo@aut.bme.hu, ORCID: 0000-0002-9070-9026

[c] E-mail: peter.ekler@aut.bme.hu, ORCID: 0000-0002-2396-3606

[d] Department of Networked Systems and Services, Budapest University of Technology and Economics, Hungary

[e] E-mail: levendov@hit.bme.hu, ORCID: 0000-0003-1406-442X

DOI: 10.14232/actacyb.289455

application. We also further elaborate on our novel solution, first introduced in [7]. After its brief introduction, our method is compared with the other strategies using the common mathematical framework.

The paper is structured as follows: Section 1 provides an overview of the currently used revocation schemes and their main characteristics. In Section 2, we present a detailed description of our new approach. Section 3 deals with the formal description of the performance characteristics of the different strategies, including our solution. In Section 4, we verify our cost model by measuring different revocation schemes in a real application. Finally, Section 5 wraps the discussion by providing an overview of the work done.

1.1 Literature review

The main source of literature regarding JSON Web Tokens is the *Request for Comments* (RFC) documents of the Internet Society (ISOC). For example, RFC 7519 [9] describes the basics of JSON Web Tokens (JWT) as "URL safe means of representing claims to be transferred between two parties".

While this definition is correct, JWTs are increasingly used in web applications as part of different authentication and authorization schemes, such as bearer tokens [6] in the OAuth2 framework [5] [10] or OpenID Connect [13]. The introspection of these tokens are described in RFC 7662 [12], which briefly touches the question of revocation without providing details on its implementation.

As the problem is more of a practical than theoretical kind, current industry approaches for JWT revocation can be found in technical documentation of different authentication solutions [11] [2], or technical blog posts such as [4] instead of more traditional scientific papers.

1.2 Overview of JSON Web Token Revocation methods

A JWT used to determine access for a protected resource is called an access token in these schemes. The token is usually digitally signed or otherwise cryptographically secured [8]. In both cases, we simply refer to the signing key or the public key as a 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. This revocation is not a trivial task, as the validity of a JWT is determined by the cryptographic assurance, which cannot be easily revoked.

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 (i.e. the refresh token is destroyed). Hence when the short lifespan runs out, no further access is possible to the system.

Blacklist: In the case of a blacklist, revoked access 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 token's validity 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 novel revocation algorithm

Our novel revocation strategy is based on the extension of the third option, which is based on 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. If a user logs out, every other user in the system must request a new token. In practice, this method scales very poorly with the increased number of clients as the number of revocation events, and thus new token acquisitions increase exponentially.

The basic idea of our method is to control the number of revocation events 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 logouts are typically infrequent events, one can use statistical methods as described in [7] to calculate an optimal group size, which minimizes the number of unnecessary revocations while maintaining a manageable number of secrets.

Being able to control the group sizes, we can optimize the performance of the strategy for our specific system. Using a larger number of client groups means less token revocations (network and computation overhead) in exchange for more memory usage.

A typical system using our strategy consists of 3 types of components; a *secured service* providing services to *clients* and an *auth server* storing authentication information and handling token generation. The process described in the following is demonstrated in more detail in Figure 1.

1. Initial token acquisition is very similar to other methods; the client authenticates themselves e.g., by giving a username/password combination.

2. After successful authentication, they are provided with an access and a refresh token. The access token is a JWT, used to consume the secured service, while the refresh token is used to acquire new access tokens.

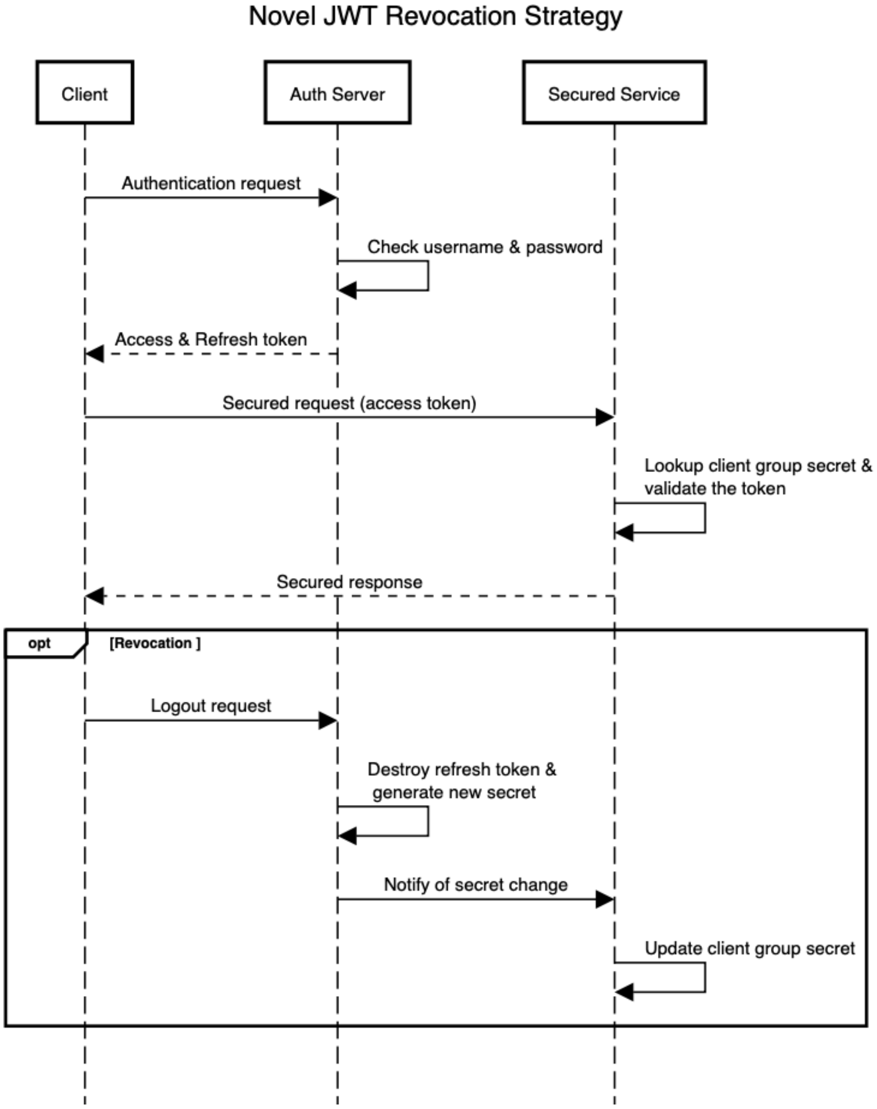
Novel JWT Revocation Strategy
Client
Auth Server
Secured Service
Authentication request
Check username & password
Access & Refresh token
Secured request (access token)
Lookup client group secret & validate the token
Secured response
opt
[Revocation]
Logout request
Destroy refresh token & generate new secret
Notify of secret change
Update client group secret

Figure 1: Detailed call structure of our novel strategy

3. Clients use the access token to consume the secured resources. Upon such a request, the secured service looks up the secret used for the given client group and validates the token against it.

4. When a client wants to log out, they signal the auth server, which destroys the refresh token, changes the group secret, and notifies the connected services.

5. Upon receiving an access token signed with the older secret, the secured service refuses the request. If the initiating client still has a valid refresh token, they can acquire a new access token with the new secret and continue using the system.

2.2 Propagating secret change events

With our proposed 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 where such a channel is unavailable, the secret change events can be propagated by the tokens themselves, albeit with lower security guarantees. In this approach, the JWT Secret is generated as a rolling code by a pseudo-random number generator [3], 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 (the 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).

3 Performance evaluation framework

To predict and compare the performance of different strategies, a common mathematical framework must be set. In this section, we will describe the basic operations associated with the handling of tokens, and then determine the cost function of each revocation strategy based on these costs.

3.1 Basic operations with tokens

Regarding the token operations, we determined a set of basic operations, which make up each approach. The costs of these operations can be parametrized for the actual system, which can be based on measurements or estimations. The following main costs were 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 strategy in 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 main factor in determining the total performance cost a system must endure is determined by the clients consuming its service. By knowing their numbers and characterizing the client sessions, predictions can be made about their impact on the system [1]. In order to calculate the former, the following properties must be known about the clients. These can also be acquired by measurements or estimations based on known properties.

- 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.

For strategies based on changing the secret, the average time between token revocations - and hence secret changes - can be calculated for a group of clients based on these metrics. It is basically the expected value of the time until the first token revocation in the group, and denoted as T_{rvk}.

Knowing both the cost and occurrences of typical operations in the system, a cost function can be constructed which describes the average cost of a revocation strategy.

3.2 Cost functions of the different strategies

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 longer it is set, the less secure the system is.

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.

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

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, which makes this approach the worst in terms of scaling when increasing the 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.

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

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, it is purely determined by the age of the token itself, while in the second case, every client logout event triggers it.

Our novel strategy

As previously stated, our method works by dividing the clients into different groups, let K denote the number of these groups. As our method builds on the secret change event, the cost function is similar too, but because of the separation 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 Cost model measurement and validation

To validate our model, a test system was implemented, and different measurements were run. This section describes the measurement setup and the results achieved.

4.1 Setup

The test setup is a set of applications written to simulate the working of a real system. The following components were implemented.

- An **Auth server**, which implements the different revocation schemes, provides tokens and stores users.

- **Foo service**, which is an example of a service providing protected resources.

- **Bar service**, which is another example of a protected service.
- **Test client**, which emulates the behavior of one or more clients. This component gathers and aggregates the data during the measurement.

The source code can be found at the following link: `https://github.com/jlaci/JWT-Revocation-Test`. The components were written in Java 11 (Zulu Open JDK 11.0.5), using the Spring Framework (managed by Spring Boot 2.3.1), the backing database was a MySQL 8. The hardware running the test was a 2019 MacBook Pro 16"(Intel Core i9-9880H, 16GB DDR4 at 2667 MHz, 1TB SSD).

The system can be run with different profiles, each corresponding to the various revocation schemes. The length of the measurement, the number of clients, and their characteristics can be configured, while cost values are based on the actual system implementation.

These system costs were measured before running the simulation by executing the associated operation 500 times and averaging the results. After that, each revocation scheme was measured with the same characteristics, and the results were aggregated.

4.2 Results

First, the actual values are determined for each cost. Table 1 lists the measured values of the different cost model terms.

Table 1: Measured values of each costs

Cost	Value(ms)
C_i (Issue cost)	28.13
C_v (Validation cost)	0.17
C_d (Communication cost)	0.51
C_c (Data access cost)	4.276

Comparison of the different schemes in a typical system

The first simulation was set up to represent a typical system employing the different JWT revocation schemes. This run aims to validate the outlines of our mathematical model and give an overview of how each strategy would fare in a close-to-real application. The simulation parameter values were chosen based on the author's experience with typical systems to represent a realistic application without favoring any strategy.

This simulation was run for 300 seconds with 100 clients. Client session characteristics were chosen to follow a uniform distribution with a mean value of 60 seconds and a maximum of 120 seconds. With 100 clients, this means T_{rvk} was calculated as 1.2 seconds. Each client, on average, executed 0.5 authenticated requests per second. Our novel algorithm used 20 client groups ($K = 20$).

Table 2 show the raw numerical results for the predicted and actual costs of each revocation strategy given the same system parameters.

Table 2: Predicted and measured cost

Strategy	Predicted	Measured
Short-Lived	0.10	0.22
Blacklist	0.22	0.41
Secret Change	2.35	3.17
Novel	0.13	0.45

Figure 2 visualizes the predicted and measured values for better understanding. As we can see from the results, the actual, measured costs are always higher than our predicted ones. This difference can be attributed to factors that were not modeled in our framework, e.g., technical overheads such as serialization and deserialization, logging, and generic costs of serving requests in a thread-pool based model. These costs are not primarily associated with the JWT revocation strategy itself; including them would make the model impractically complex.

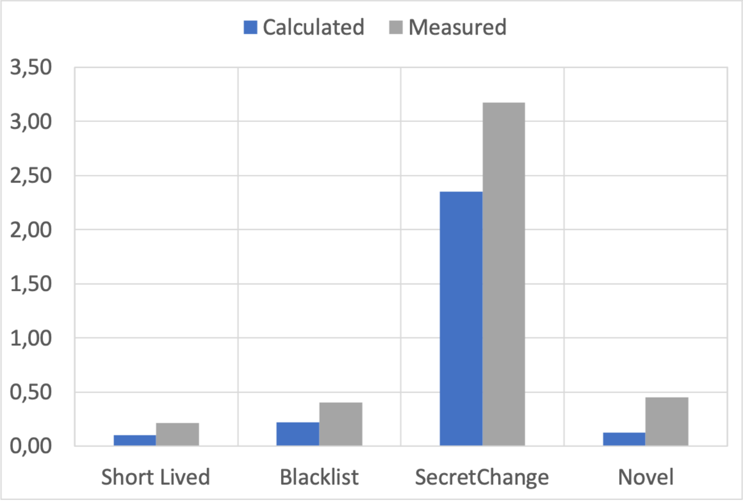

Figure 2: The comparison of results

Our expectations regarding the scale and relative performance of each strategy were mostly met. The largest error between measurements and prediction occurred with our novel approach, most likely due to implementation constraints, such as the library we used for encryption and decryption not being optimized for frequent secret changes. Nevertheless, even with this less accurate measurement, we can see the following general tendencies. The short-lived method provides the best performance while offering the worst security. Blacklist and our novel approach have roughly the same performance for $r = 0.5$ request / client / second with the same level of security, while secret change has the worse performance with slightly worse security.

Comparing blacklist and novel approach for larger load

Based on the previous data, the blacklist and our solution would appear to be the same in terms of performance for relatively low request numbers. This changes, however, when increasing the number of requests per second made by the clients. As seen in Figure 3 with the increased number of requests, our solution scales much better than the blacklist approach while providing the same level of security.

Due to processing power constraints of the test setup, we could not measure beyond this point $r = 2.5$ with a reasonable number of clients. If we set the target r higher than this, the actual measured rate of requests were not increased accordingly, due to resource contention between the simulation threads. Based on the calculations and previous measurements, however, we are confident that the trend would continue, and our solution would prove to be better than any other strategy (blacklist and secret change) with the same security level.

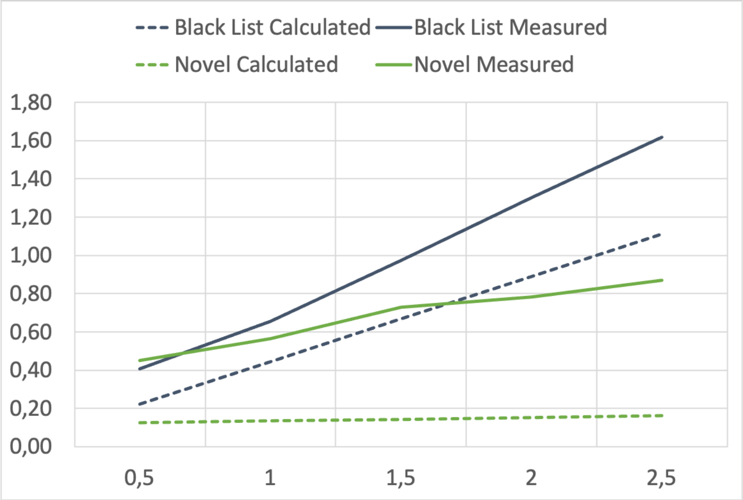

Figure 3: The comparison black list and novel

5 Conclusions

In this paper, we described the main approaches of JWT revocation and introduced our novel solution. We provided a tool-set for choosing the best revocation strategy based on the different characteristics of the system. The basis of this tool-set is the measurement of client population characteristics and costs associated with common operations when dealing with JWTs.

We proved the validity of our cost model by implementing a system using the described methods and measuring its performance for different revocation strategies. The measured results supported that our novel revocation method is competitive with the current solutions while offering advantages in usability, such as instantaneous revocation. We showed that our solution scales better for an increased number of requests than any other strategy with the same security guarantees (instantaneous revocation).

With the proven mathematical framework at hand, one can find the optimal revocation strategy 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 the capacity planning and system design of distributed systems, as the JWT based authentication schemes have the highest potential in this area. We hope that our novel solution will help with increasing the performance and efficiency of these systems, therefore conserving energy and costs associated with providing a large-scale distributed service.

References

[1] Arlitt, Martin. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):50–63, 2000. DOI: `10.1145/362883.362920`.

[2] Auth0 Inc. Revoke tokens. `https://auth0.com/docs/`.

[3] Blum, Manuel and Micali, Silvio. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing*, 13(4):850–864, 1984. DOI: `10.1137/0213053`.

[4] dWTV. Learn how to revoke JSON Web Tokens, July 2017. `https://developer.ibm.com/tv/learn-how-to-revoke-json-web-tokens/`.

[5] Hardt, Dick. The OAuth 2.0 authorization framework. Technical report, RFC 6749, October, 2012.

[6] Hardt, Dick and Jones, Michael. The OAuth 2.0 authorization framework: Bearer token usage. `https://tools.ietf.org/html/rfc6750`.

[7] Jánoky, László Viktor, Levendovszky, János, and Ekler, Péter. An analysis on the revoking mechanisms for JSON Web Tokens. *International*

Journal of Distributed Sensor Networks, 14(9), September 2018. DOI: 10.1177/1550147718801535.

[8] Jones, M., Bradley, J., and Sakimura, N. JSON Web Signature (JWS). RFC 7515, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7515.txt.

[9] Jones, M., Bradley, J., and Sakimura, N. JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.

[10] Okta Inc. Self-encoded access tokens. https://www.oauth.com/oauth2-servers/access-tokens/self-encoded-access-tokens/.

[11] Pontarelli, Brian. Revoking JWTs & JWT expiration. https://fusionauth.io/learn/expert-advice/tokens/revoking-jwts/.

[12] Richer, Justin. OAuth 2.0 token introspection. https://tools.ietf.org/html/rfc7662.

[13] Sakimura, Natsuhiko, Bradley, John, Jones, Mike, De Medeiros, Breno, and Mortimore, Chuck. OpenID connect core 1.0. *The OpenID Foundation*, page S3, 2014.