Risk of Information Loss Using JWT Token

Oleksandra Bulgakova[1], Viktor Mashkov[2], Viacheslav Zosimov[1] and Pavlo Popravkin[1]

[1]*V.O.Sukhomlynsky National University of Mykolaiv, Nikolska 24, Mykolaiv, 54000, Ukraine*
[2]*Jan Evangelisty Purkynė University, Ceske mladeze, 8, Usti nad Labem, 40096, Czech Republic*

Abstract

This paper presents a variant of saving the JWT token, which allows you to protect yourself from typical attacks on the server using an access token, which significantly reduces risks and simplifies compliance with some requirements of security standards. Shows how the web key interacts with the server, which provides protection against server-side attacks such as XSS and CSRF. The interaction of the JSON web key with the server and the solution of the main problems of authorization and storage of JWT - the JSON web token are considered.

The experimental results show the advantage of the proposed method ("quiet" token) over using the storage of the JWT token in LocalStorage, cookie when accessing the token during XSS and CSRF attacks.

The proposed method for storing the JWT token to protect data from typical attacks on the server. The lack of a token in local storage has the advantage of persisting data as the keys are no longer available information.

Keywords

Structured information, web key, token, cookie, CSRF attack, XSS attack, risk of information loss

1. Introduction

One of the most important and complex tasks in information security is the protection of confidential data. In the case of confidential data, a variety of solutions using encryption technologies are usually used, but in recent years, there has been an increase in interest in another technology - tokenization, which reduces the risks for confidential data. Applications can store, use and make transactions using only a token and without putting real data at risk. Although some operations require access to real data, tokenization minimizes their use. The key advantage of tokenization is the ability to store confidential data in only one place - on the tokenization server, where it is securely stored in encrypted form. This reduces the risk compared to encryption, which makes sensitive data available in multiple locations [1].

CITRisk'2021: 2nd International Workshop on Computational & Information Technologies for Risk-Informed Systems, September 16–17, 2021, Kherson, Ukraine
EMAIL: sashabulgakova2@gmail.com (O.Bulgakova); viktor.maskov@ujep.cz (V.Mashkov); zosimovvv@gmail.com (V.Zosimov); pavel.popravkin.dm@gmail.com (P.Popravkin)
ORCID: 0000-0002-6587-8573 (O.Bulgakova); 0000-0001-9817-3388 (V.Mashkov); 0000-0003-0824-4168 (V.Zosimov); 0000-0002-6731-7016 (P.Popravkin)

That is why the paper proposes a variant of saving the JWT token, which allows you to protect yourself from typical attacks on the server using an access token, which significantly reduces risks and simplifies compliance with some requirements of security standards.

JSON Web Token determines the particular structure of information that is sent over the network. It is presented in two forms - serialized and deserialized. The first is used directly to transfer data with requests and responses. The other one reads and writes information to the token, that is, the deserialization process is performed [2]. This article has covered the persistence of a serialized form, which typically happens in cookies or browser local storage. Local Storage - the method is dangerous because it is exposed to attacks such as XSS [3]. Cookie is a simple storage of a token, often threatens with a CSRF attack and does not protect against XSS attacks [4-5].

The proposed token saving approach allows one to protect against the two most frequent types of attacks and to organize a more reliable security algorithm. Each of these methods has drawbacks that can be avoided by storing the token in a local variable inside the closure.

2. Types of Attacks

XSS (Cross-Site Scripting) is a vulnerability that an attacker can inject into a page through JavaScript code. This code will be executed every time users visit the application page where this code was added. With this code, an attacker can get the user authorization information and enter in his account, or redirect the user to another page, clone, etc. - almost anything JavaScript can do is made available to an attacker. Comparing XSS with SQL injection, XSS is safe for the server, but poses a threat to the users of the infected resource. However, if an administrator cookie gets to an attacker, you can gain access to the control panel of the site and its contents [3].

CSRF (Cross-Site Request Forgery, also XSRF). The essence of CSRF is that browsers don`t understand how to distinguish whether an action was explicitly performed by the user (such as clicking a button on a form or following a link) or whether the user unintentionally performed the action. An effective and generally accepted method of protecting against CSRF-Attacks today is a token - a random set of bytes that the server transmits to the client, and the client returns to the server [4].

3. Saving a Token

3.1. Saving a token in a cookie

The main advantage of storing a token in a cookie is that the tokens are not available with JavaScript. As a result, the vulnerability to XSS attacks is much lower than for local repositories but also does not fully protect against XSS attacks.

If you use the HttpOnly flag and secure cookies, this means that JavaScript cannot be accessed from these files. That is, even if an attacker can run his code on this page, you will not be able to read the cookie access token.

Cookies are automatically sent in each HTTP-requests (Hypertext Transfer Protocol - HTTP) to the server.

The problem with storing a JSON web token in a cookie is a vulnerability to CSRF attacks. Also, depending on the specific circumstances, it may happen that the tokens in the cookie can’t be saved.

The size of cookies is limited to 4 KB. Therefore, to use large JWTs, storing them in a cookie will not work [6-7].

There are scenarios in the implementation of which you can`t send cookies to your API-server (Application Programming Interface - API). It is also possible that some API requires a token to be placed in the Authorization header. In this case, it is impossible to store tokens in cookies.

Context-sensitive source coding is commonly used to prevent XSS vulnerabilities. In some cases, this may be sufficient to encode special HTML characters (Hypertext Markup Language - HTML), such as opening and closing tags. References are usually prohibited unless you start with a whitelist, such as http:// or https://, which prevents the use of URI schemes (Uniform Resource Identifier - URIs), such as javascript://.

It should also be noted that most modern web browsers have a built-in XSS filter, this protection should not be considered as a warranty. It can't catch all kinds of cross-site scripting attacks and isn't very smart, which can sometimes lead to false positives that can prevent some pages from loading. The web browser's XSS filter should only be the “second line of defense”.

Sometimes developers remove dangerous functions and characters from the code. This solution is fundamentally incorrect because XSS browser filters cannot recognize dangerous useful data when the original data is forged given possible workarounds.

3.2. Saving a token in LocalStorage

The main advantage of local storage is that they are easy to use.

Working with local storage is very convenient, it uses JavaScript. If the program does not have a backend, then relying on other people's APIs, it is not always possible to track whether these APIs use personal data from cookies on this site.

Using LocalStorage, it is convenient to work with APIs that must find an access token in the request header. Example:Authorization Bearer $ {access_token}.

The main disadvantage of local storage is the vulnerability to XSS attacks.

When performing an XSS attack, an attacker could run dangerous JavaScript code on your site. This means that the attacker can access the access token stored in localStorage.

The source of the XSS attack may be third-party JavaScript code included in this site. It can be something like React, Vue, jQuery, Google Analytics script, and so on. In modern conditions, it is almost impossible to develop a site that does not include libraries of third-party developers.

The main methods of protection against XSS-attacks by which developers try to protect data:

1. Try simply not to store sensitive data in localStorage, including JWT or any other credentials in this regard.
2. Use a cookie header on top of the authorization header.
3. Set cookie header protection.
4. Avoid displaying the token on the screen, in URLs (Uniform Resource Locator - URL), or the source code.

These precautionary methods do not solve the root problem of accessing sensitive data in a very light format. Storing data in LocalStorage will always be dangerous because it is a format of LocalStorage.

4. Saving Token In a Local Variable

When the token is saved in a local variable inside the closure, the system becomes protected from typical attacks - CSRF (cannot be automatically sent with cookies, because the token is stored in memory and sent as a header on every request to the server) and XSS (because the token is not saved in Session / LocalStorage).

Figure 1 shows a new session where you can get a new token and store it in a local variable inside the closure.

Figure 2 an example of obtaining a token in an XSS attack is shown. The example shows that the information is not saved in LocalStorage so an attacker can't access the file. This also applies to CSRF attacks.

Further, with each request, it is necessary to add a token to the header and avoid other storages in which there is unprotected data.

```
flight inMemoryToken;


function login ({jwt_token, jwt_token_expiry}, noRedirect) {
        inMemoryToken = {
                token: jwt_token,
                expiry: jwt_token_expiry
        };
        if (! noRedirect) {
                Router.push ('/ app')
        }
}
```

Figure 1: Example of getting a token and store it in a local variable inside the closure

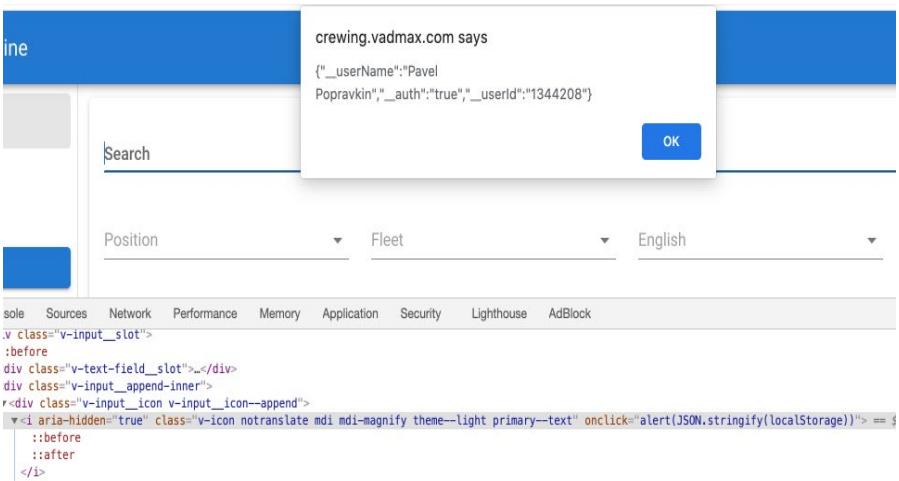

Figure 2: Getting data from localStorage in XSS attack

5. Access Problems and Their Solutions

When using the application, the user may encounter some problems: a quick end of the session (the JWT expires) and the session will not be saved upon re-entry (the system does not save the JWT token on the user's side). To solve these problems, you can give a refresh token that can be used for the API and that can be saved between user sessions.

This token is part of the authentication process along with the JWT. The server stores the refresh token and associates it with a specific user in its database. On the client-server, you need to connect the application to create the update and get a new JWT before the previous JWT token expires.

The refresh token is sent as HttpOnly and is automatically sent by the browser when using the API.

Figure 3 shows a new login process where the refresh token will be sent along with the JWT.

User input string through the token update will take place in four stages:

1. The user logs in through the API.
2. The server generates JWT tokens and updates.
3. The server sets an HttpOnly cookie with a refresh token and returns to the user as JSON. JWT is stored in memory.
4. Based on the expiration of the JWT, a “silent” token renewal is triggered.

Figure 4 shows a silent update that takes place in three stages:

1. The endpoint of the call is the refresh token.
2. The server reads the httpOnly cookie and returns a new JWT.
3. Sets a new refresh token cookie through the Set-Cookie header.

If the user has logged out of the current session, then when he logs into the application again, the system will look like in Figure 5

Thus, it becomes possible to support user authorization when the token expires.

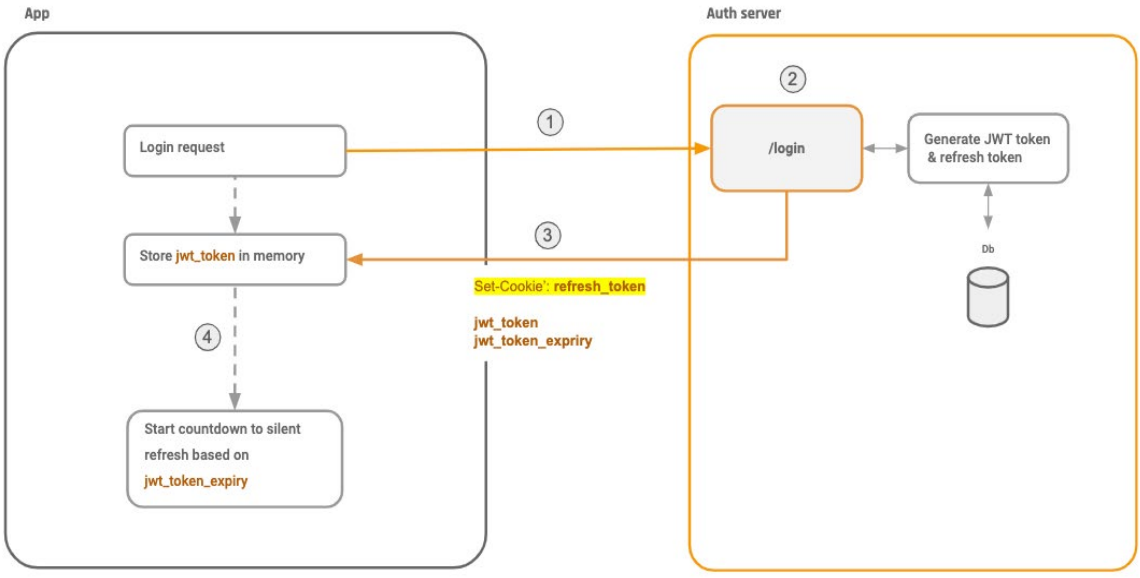

Figure 3: Update token sheme

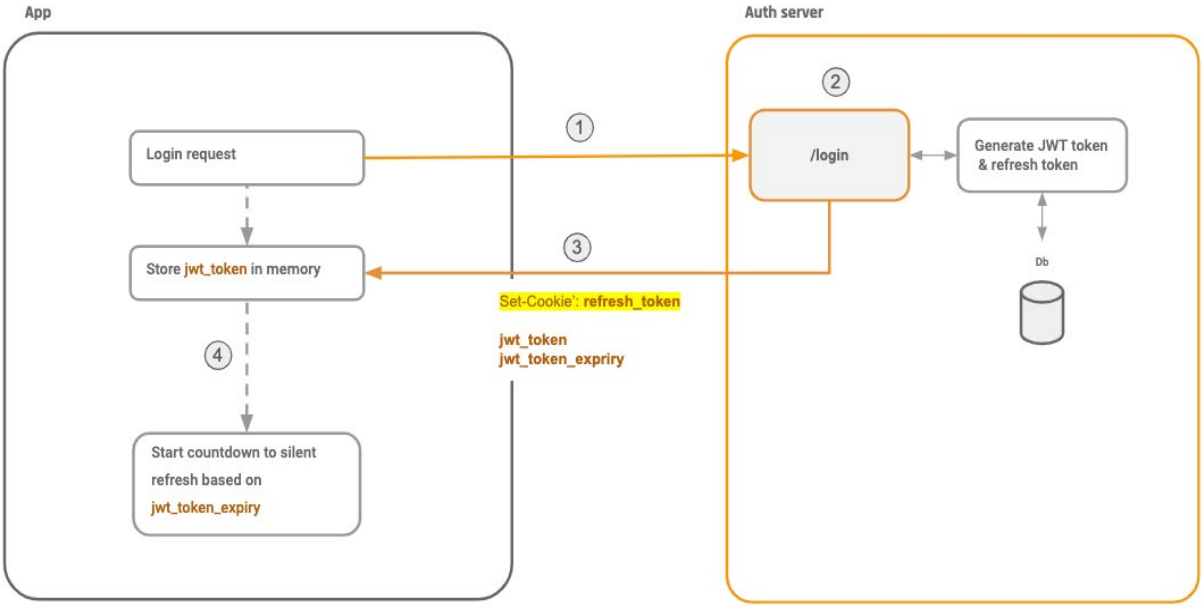

Figure 4: Silent update sheme

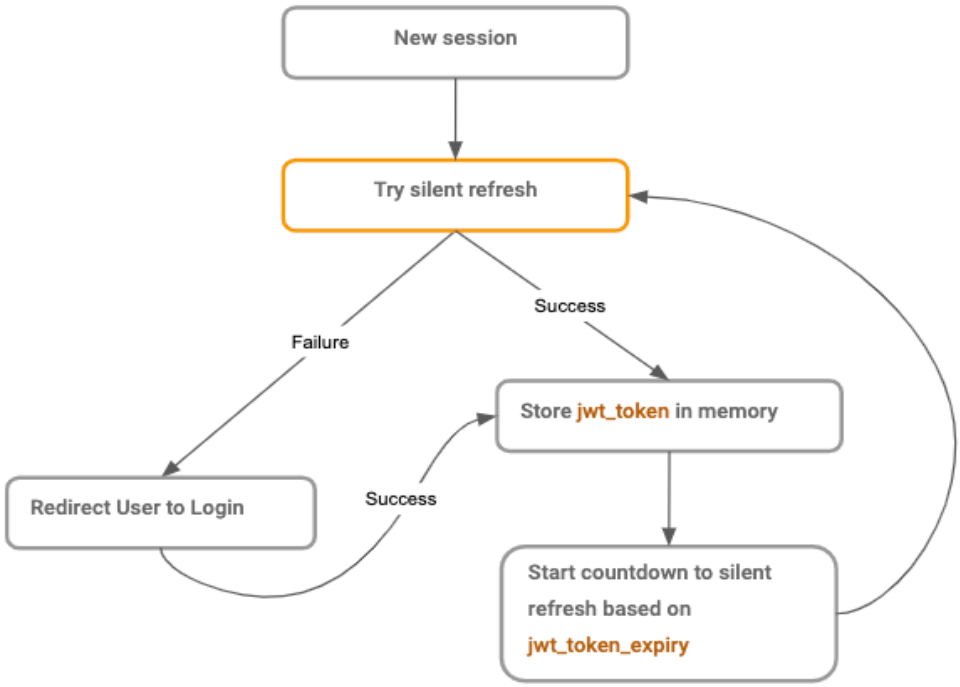

Figure 5: Workflow scheme when user logged out

Table 1 presents the results of experiments that show the possibility of gaining access to the token during XSS and CSRF attacks using the storage of the JWT token in LocalStorage, the cookie and the proposed method (“silent” token). A simulation of 30 experiments was carried out. “+” means that it was possible to get access, “-” access was not obtained.

Table 1

The result of using the considered methods for various attacks

Method	XSS attack (30)	CSRF attack (30)
LocalStorage	+ (23 +; 7-)	- (8 +; 22-)
cookie	- (9 +; 21-)	+(26 +; 4-)
“silent” token	- (2 +; 28-)	-(1 +; 29-)

Out of 30 test situations (Table 1), the proposed method ("quiet" token) turned out to be the most effective: with XSS attacks only 2 out of 30, and CSRF attacks - only 1 out of 30 attempts to obtain a token.

6. Conclusions

Cybercrime is growing every year and methods of attack are becoming more sophisticated. All this should contribute to the development of methods and policies of protection. With the increasing use of brute force attacks and phishing attacks to capture user data, it is becoming clear that password authentication is no longer sufficient to counter attackers.

Token-based authentication, when used in conjunction with other authentication methods, creates a barrier designed to stop even the most advanced hacker. Token authorization systems are considered to be very secure and efficient, but despite the many benefits associated with using tokens, there is always a small risk of losing confidential information.

That's why this paper discusses variant of saving the JWT token, which allows you to protect yourself from typical attacks on the server using an access token, which significantly reduces risks and simplifies compliance with some requirements of security standards. The risk of information loss using JWT token of storing structured information in local storage that is sent over the network in a serialized form, which usually happens in cookies or browser local storage. Local storage - the method is dangerous because it is susceptible to attacks such as XSS. A cookie is a simple token store that is often threatened by CSRF attacks and does not protect against XSS attacks.

This article proposes a method for storing a JWT token to protect data from typical server attacks. The lack of a token in local storage has the advantage of persisting data as the keys are no longer available information. The proposed token saving approach allows one to protect against the two most frequent types of attacks and to organize a more reliable security algorithm. Each of these methods has drawbacks that can be avoided by storing the token in a local variable inside the closure.

During the experiment, various situations were generated to gain access to the token. Out of 30 test situations, the proposed method ("quiet" token) turned out to be the most effective: with XSS attacks only 2 out of 30, and CSRF attacks - only 1 out of 30 attempts to obtain a token.

However, you need to understand that this method does not provide a complete guarantee of protecting the system from attacks - it excludes only some of the types of attacks described above.

The JWT token storage method proposed in this article is experimental, and work and research is currently underway to improve it for use in other types of attacks.

References

[1] Understanding and Selecting a Tokenization Solution, 2020. URL: https://securosis.com

[2] JSON Web Tokens, 2021. URL: https:// jwt.io

[3] Password stealing from HTTPS login page and CSRF protection bypass with reflected XSS, 2020. URL: https:// medium.com/@MichaelKoczwara/password-stealing-from-https-login-page-and-csrf-bypass-with-reflected-xss-76f56ebc4516
Cross-Site Request Forgery Prevention Cheat Sheet, 2021. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[4] V.Zosimov, O.Khrystodorov, O.Bulgakova, Dynamically changing user interfaces: software solutions based on automatically collected user information, Programming and Computer Software, 2018, Vol 44 (6), pp. 492-498. doi:10.1134/S036176881806018X

[5] XSS and CSRF attack, 2020. URL: https:// geekbrit.org

[6] V.Jánoky, J.Levendovszky, P.Ekler, An analysis on the revoking mechanisms for JSON Web Tokens, Next Generation Internet of Things (IoT) and Cloud Security Solutions, 2018, Vol 14 (9). doi:10.1177/1550147718801535