DOI https://doi.org/10.15407/csc.2020.06.055
UDC 004.9

O.S, BULGAKOVA, PhD Ehg. Sciences, Associate Professor of the Information Technologies department, V.O. Sukhomlynsky Mykolaiv National University, Nikolska str., 24, Mykolaiv, 54000, Ukraine, sashabulgakova2@gmail.com

V.V. ZOSIMOV, Doctor of Science, Head of the Department of Information Technologies, V.O. Sukhomlynsky Mykolaiv National University, Nikolska str., 24, Mykolaiv, 54000, Ukraine, zosimovvv@gmail.com

P.D, POPRAVKIN, Master's student, specialty 122 Computer Science, V.O. Sukhomlynsky Mykolaiv National University, Nikolska str., 24, Mykolaiv, 54000, Ukraine, pavel.popravkin.dm@gmail.com

STORING JWT TOKEN IN LOCAL VARIABLES

The article discusses the problem of storing structured information over the Internet (JSON format) in local storage and pieces of information transmitted to the browser from the site visited by the user (cookies), and a method is proposed for storing the JSON web key in a local variable inside the closure (functions that refer to into independent variables). Based on user authorization, the interaction of the JSON web key with the server is shown, and the solution to the main problems of authorization and storage of the token JWT (JSON Web Token).

Keywords: *JWT, token saving, local variable, Cookie, LocalStorage, CSRF attack, XSS attac.*

Introduction

JWT (JSON Web Token) is a standard based on the JSON format that allows you to create access tokens, with the help of tokens that are usually authenticated in client-server applications [1]. When using JWT tokens, the question arises as to how to safely store tokens in the frontend part of the program. This issue must be resolved immediately after the token is generated on the server and transferred to the client part of the program. Because improper storage of tokens often leads to loss of confidential user data and hacking.

There are several options for saving the token, they all have their methods of protection against attacks but remain vulnerable to the basic CSRF (Cross-Site Request Forgery – SRF) and XSS (Cross-Site Scripting – XSS) attacks. The article aims to show the possibility of using a local variable as a repository of JWT tokens to reduce the risk of system breakage due to CSRF and XSS attacks

Problems of Saving a JWT Token

The JSON web token is typically used to transmit data for authentication in client-server applications. Tokens are created by the server, signed with a secret key, and transmitted to the client, who then uses the token to confirm his identity [1].

The JSON web token defines a special structure of information that is sent over the network. This structure is presented in two forms — serialized

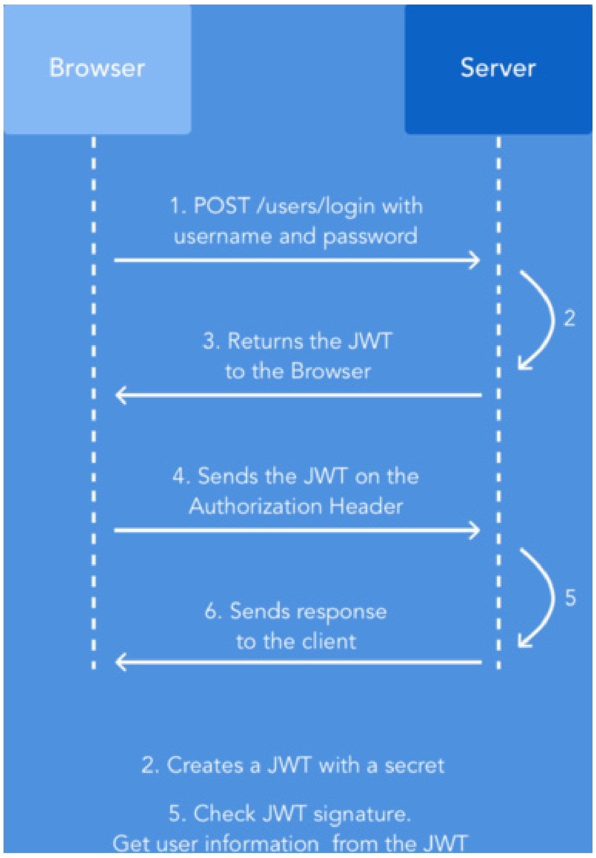
Browser
Server
1. POST /users/login with username and password
2
3. Returns the JWT to the Browser
4. Sends the JWT on the Authorization Header
5
6. Sends response to the client
2. Creates a JWT with a secret
5. Check JWT signature. Get user information from the JWT

***Fig.* 1.** Authentication mechanism

and deserialized. The first is used directly to transmit data with requests and responses. On the other hand, to read and write information in a token, you need to deserialize it.

In the non-serialized form, the JSON web token consists of a header and a payload, which are ordinary JSON objects.

Example [1, 2]: {“alg”: “HS256”} {“sub”: “user123”, “productIds”}

The JSON web token in serialized form is a string of the following format: eyJhbGciOiJub25lIn0.eyJzdWIiOiJ1c2VyMTIzIiwicHJvZHVjdElkcyI6WzEsMl19.

The serialized form is usually saved in cookies or local browser storage (LocalStorage/Session Storage). Each method has drawbacks that can be eliminated by storing the token in a local variable inside the circuit.

Local Storage/Session Storage is a dangerous method because it is vulnerable to XSS attacks. A big danger arises if you connect scripts from third-party CDN (Content Delivery Network – CDN). Also, a common problem is the lack of guarantee that the connected scripts do not send data from storage on the server-side. Moreover, if Local Storage is available between tabs in the browser, then Session Storage is only available in one tab, and opening a site in a new tab will only trigger a new round of authorization.

Cookie — simply storing an access token in a cookie often threatens a CSRF attack. Moreover, it does not protect against XSS attacks.

The new approach to saving the token allows to protect yourself from the two most common types of attacks and organize a better security algorithm.

Variety of Attacks

XSS (cross-site scripting) — a type of attack on a web system that implements malicious code on a specific page of the site and interacts with a remote attacker’s server when the user opens the page.

Local memory is vulnerable to XSS attacks because it is very easy to work with using JavaScript. Therefore, an attacker can access the token and use it to their advantage. However, although the HttpOnly cookie is inaccessible with JavaScript, this does not mean that the token using the cookie is protected from XSS attacks [3].

If an attacker can run his JavaScript code in this application, it means that the hacker can simply send a request from this server, and the token will be included in this request automatically. This scheme is simply not so convenient for the attacker, as the attacker cannot read the contents of the token. Also, with this scheme, hackers may find it more profitable to attack the server using the victim’s computer rather than their own [4].

Cross-site request forgery (CSRF) is an attack that forces an end-user to perform unwanted actions on a web application in which the application is currently authenticated. An attacker fraudulently forces a user of a web application to perform actions to select an attacker. If the victim is a regular user, a successful CSRF attack can force the user to make requests for status changes, such as money transfers, email changes, etc. If the victim is an administrator account, the CSRF can compromise the entire web application.

An attack with forged cross-site requests consists of two main parts. The first is to trick the victim into opening a link or loading a page. This is usually done through social engineering and malicious links. The second part is sending a processed, legitimate form of a request from the victim's browser to the website. The request is sent with the values selected by the attacker, including any cookies that the victim has linked to this website. This way, the website knows that this victim can perform certain actions on the website. Any request sent with these HTTP credentials or cookies will be considered legitimate, even if the victim submits the request at the attacker's command [5].

When a website is requested, the victim's browser checks to see, if it has any cookies related to the source of that website that needs to be sent with the HTTP request. If so, these cookies are included in all requests sent to this website. The cookie value usually contains authentication data, and such cookies represent the user's session. This is done to provide the user with a convenient operation, so it does not require re-authentication for each page visited. If the website approves the session cookie and believes that the user's session is still valid, the attacker can use CSRF to send requests as if they were sent by the victim. The website cannot distinguish between requests sent by an attacker and requests sent by a victim, because requests are always sent from the victim's browser with their cookie. The CSRF attack simply takes advantage of the fact that the browser automatically sends a cookie to the website with each request.

Forgery of cross-site requests will be effective only if the victim is authenticated. This means that the victim must log in for a successful attack. Because CSRF attacks are used to bypass the authentication process, there may be some elements that are not affected by these attacks, or even these elements are not protected from them. For example, a site accepts requests to change your email address:

```
POST/email / change HTTP / 1.1

Host: site.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 50

Cookie: session = abcdefghijklmnopqrstu


email = myemail.example.com
```

In this situation, a POST request is sent to https://site.com/email/change. In this case, session cookies will be automatically included in this request.

You can protect against this threat by using the SameSite attribute in the response header and anti-CSRF tokens. But this process also has some limitations [6].

Token Storage in a Local Variable

Advantages over storing the token in LocalStorage and cookies. Storing a token in a local variable inside a circuit has data protection advantages over storing a token in LocalStorage and a cookie. First, the system becomes completely protected from typical CSRF and XSS attacks. A token in memory cannot be received during an XSS attack because the token is not stored in LocalStorage and SessionStorage, there is also protection against CSRF attacks. After all, each token cannot be sent automatically with cookies because the token is stored in memory and its sending occurs as a header in each request to the server.

As soon as the page opens, a new session begins, after which you can get a token and save it in a local variable inside the circuit.

In an XSS attack, an attacker will also not be able to retrieve the token from an open browser window through LocalStorage, because we do not store this information there. This also applies to CSRF attacks.

In the future, each request must add a token directly to the header and avoid other repositories in

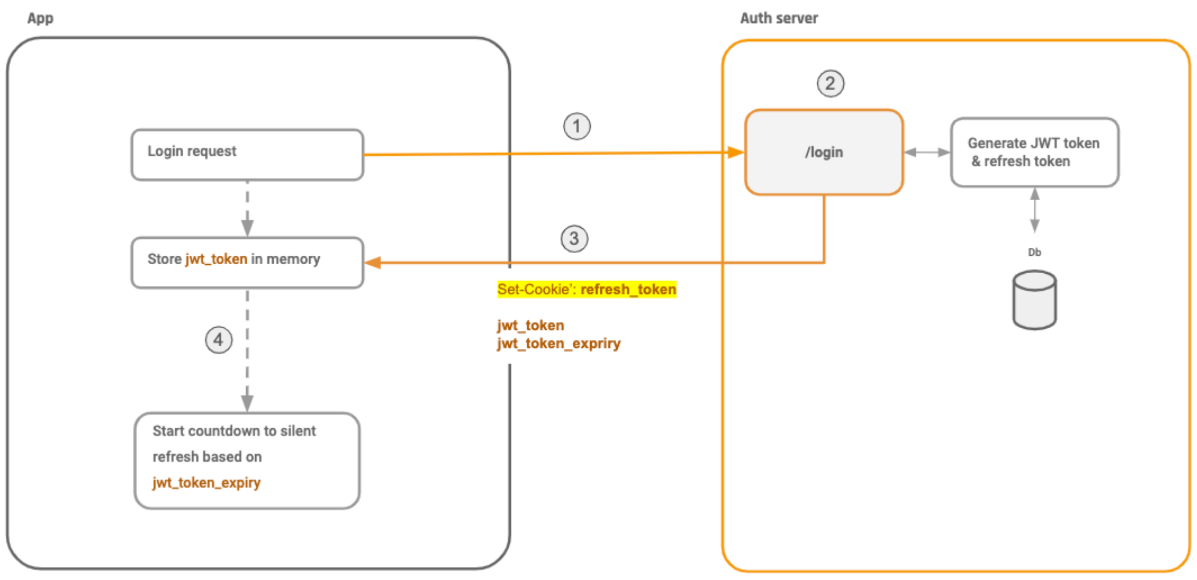
App
Auth server
1
2
3
4
Login request
/login
Generate JWT token & refresh token
Db
Store jwt_token in memory
Set-Cookie': refresh_token
jwt_token
jwt_token_expriry
Start countdown to silent refresh based on jwt_token_expiry

***Fig.* 2.** User login scheme via update token

its path that have unprotected areas in the security system.

Problems of Providing Access to the User and Their Solution

There are 2 main issues that app users will face sooner:

1. Due to the short expiration time of the JWT, the user will log out every 15 minutes.

2. If the user closes this application and reopens it, he will need to log in again. Their session is not saved because the system does not store the JWT token on the client-side.

To solve the problem, most JWT providers provide an update token (). The update token has 2 properties:

1. It can be used to call an API (for example an update token) to obtain a new JWT token before the previous JWT expires.

2. It can be safely stored between sessions on the client.

This token is issued as part of the authentication process along with the JWT. The authentication server must store this update token and associate it with a specific user in its database so that the user can process the updated JWT logic.

On the client, before the expiration of the previous JWT token, you need to connect the application to create the endpoint update token and get a new JWT.

The update token is sent by the client authentication server as an HttpOnly cookie and is automatically sent by the browser with the update token when the API is called. Because Javascript on the client-side cannot read or steal the HttpOnly cookie, this solution is better for mitigating the effects of XSS than saving it as a regular cookie or in a local repository.

This is safe from CSRF attacks because, even if a form submission attack can trigger an update token, an attacker cannot obtain the new value of the JWT token that is returned.

Note that although this method is not resistant to severe XSS attacks, it is recommended that you use the HttpOnly cookie to store session-related information in combination with conventional XSS mitigation methods. However, storing this ses-

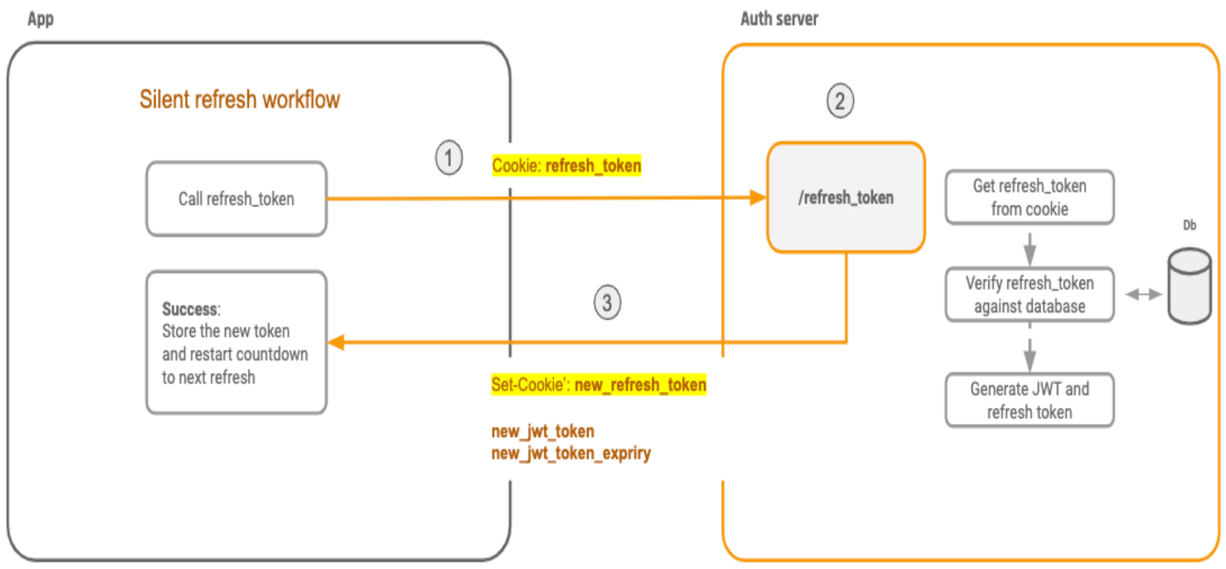

***Fig.* 3.** Silent update scheme

sion indirectly through the update token prevents direct CSRF — a vulnerability that could occur with the JWT token.

The new "login" process will now look like this: Nothing special changes, except that the update token is sent with the JWT, Fig. 2.

The sequence of user login through the update token will be carried out in 5 steps:

1. The user logs in using an API call.
2. The server generates a JWT token and an update token
3. The server sets an HttpOnly cookie with an update token. The JWT and the JWT expiration date are returned to the client as a JSON payload.
4. JWT is stored in memory.
5. The countdown to a future silent update is triggered based on the JWT expiration date.

The update now looks like this: (Fig. 3):

1. The update token is the endpoint of the call.
2. The server reads the httpOnly cookie, and if a valid update token is found, the server returns a new JWT and its expiration date to the client and sets a new update token cookie via the Set-Cookie header.

Suppose a user has logged out of the current session and closed the browser tab. Now, when the user logs into the application again, the system looks like this (Fig. 4):

1. If there is no JWT in memory, the silent update workflow starts.
2. If the update token is still valid (or has not been revoked), a new JWT is sent.

This creates an opportunity to maintain the authority of the client inadvertently at the end of the access token.

Conclusion

The article proposed an option to save the JWT token, which allows you to protect yourself from typical attacks on the server using the access token. The token cannot be obtained during XSS and CSRF attacks, as mentioned in the article, unlike

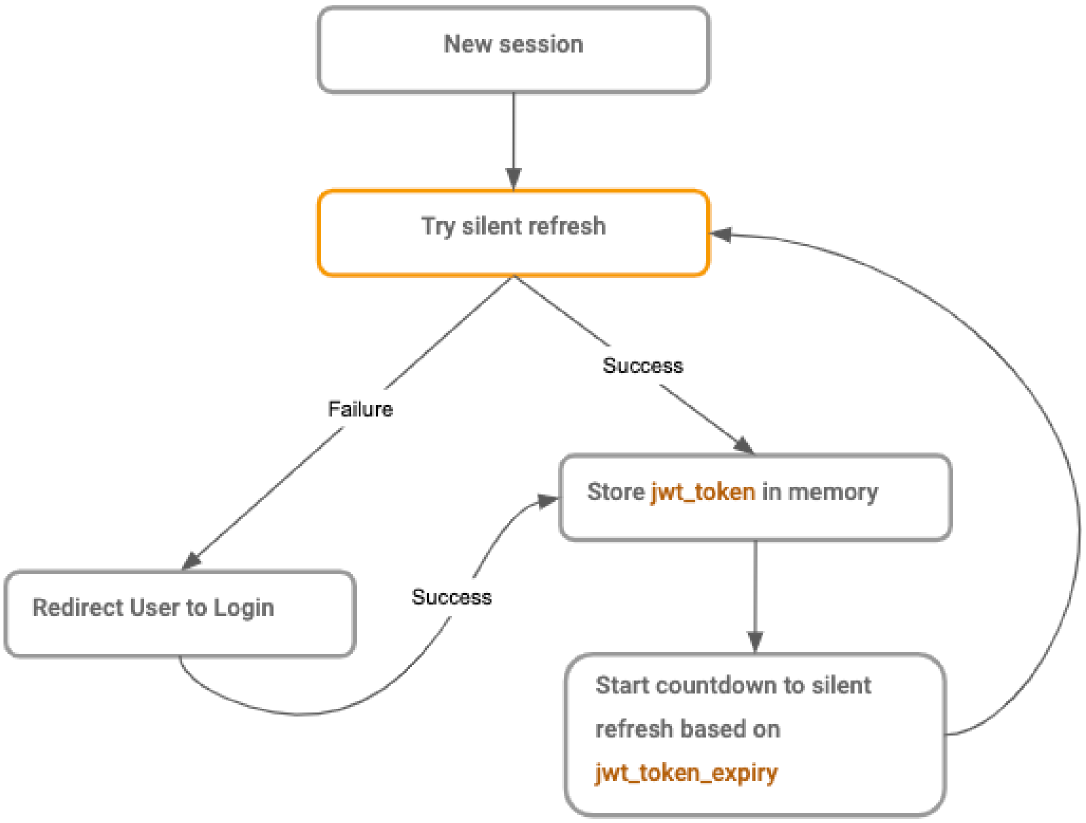
New session
Try silent refresh
Failure
Success
Store jwt_token in memory
Redirect User to Login
Success
Start countdown to silent refresh based on jwt_token_expiry

***Fig.* 4.** Workflow diagram when the user has closed the tab

when the token is stored in LocalStorage or cookie, because it is stored in memory, and its absence in local storage gives an advantage in data storage because confidential keys are no longer public. Each time an attacker tries to obtain confidential data through an access token, he will spend more time searching for and retrieving the token than other methods of storing the access token, so this method can be considered better than storing the token in LocalStorage or a cookie.

Always remember that JWT is better not to use unnecessarily, it has many patterns to consider. An error can result in the loss of sensitive data. It should also be noted that storing a JWT token in a local variable does not fully guarantee the vulnerability of this system to attacks, with this method you can exclude several variants of attacks from the list, and complicate the hacking of our system.

Data security should be a top priority when creating systems with confidential information.

REFERENCES

1. COURSE on Udacity "Scalable Microservices with Kubernetes by Google". [online] Available at: <https://www.udacity.com/course/scalable-microservices-with-kubernetes--ud615> [Accessed 11 Nov. 2020].
2. JSON Web Tokens. [online] Available at: <jwt.io> [Accessed 11 Nov. 2020].
3. Cross Site Scripting (XSS) Software Attack. [online] Available at: <https://owasp.org/www-community/attacks/xss/> [Accessed 11 Nov. 2020].
4. Cross Site Scripting Prevention Cheat Sheet. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> [Accessed 11 Nov. 2020].
5. Password stealing from HTTPS login page and CSRF protection bypass with reflected XSS. [online] Available at: <https://medium.com/@MichaelKoczwara/password-stealing-from-https-login-page-and-csrf-bypass-with-reflected-xss-76f56ebc4516> [Accessed 11 Nov. 2020].
6. Cross-Site Request Forgery Prevention Cheat Sheet. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html> [Accessed 11 Nov. 2020].

Received 27.10.2020

ЛІТЕРАТУРА

1. COURSE on Udacity "Scalable Microservices with Kubernetes by Google". URL: https://www.udacity.com/course/scalable-microservices-with-kubernetes--ud615
2. JSON Web Tokens. URL: jwt.io
3. Cross Site Scripting (XSS) Software Attack. URL: https://owasp.org/www-community/ attacks/xss/
4. Cross Site Scripting Prevention Cheat Sheet. URL: https://cheatsheetseries.owasp. org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
5. Password stealing from HTTPS login page and CSRF protection bypass with reflected XSS. URL: https://medium.com/@MichaelKoczwara/password-stealing-from-https-login-page-and-csrf-bypass-with-reflected-xss-76f56ebc4516
6. Cross-Site Request Forgery Prevention Cheat Sheet. URL https://cheatsheetseries. owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Надійшла 27.10.2020

О.С. Булгакова, кандидат технічних наук, доцент, Миколаївський
національний університет імені В.О. Сухомлинського,
54001, м. Миколаїв, вул. Нікольська, 24, Україна,
sashabulgakova2@gmail.com

В.В. Зосімов, доктор. технічних наук, доцент, завідувач кафедри,
Миколаївський національний університет імені В.О. Сухомлинського,
54001, м. Миколаїв, вул. Нікольська, 24, Україна,
zosimovvv@gmail.com

П.Д. Поправкін, студент магістратури спеціальності 122 Комп'ютерні науки,
Миколаївський національний університет імені В.О. Сухомлинського,
54001, м. Миколаїв, вул. Нікольська, 24, Україна,
pavel.popravkin.dm@gmail.com

ЗБЕРІГАННЯ *JWT* ТОКЕНА У ЛОКАЛЬНІЙ ЗМІННІЙ

Вступ. При роботі з аутентифікацією в інформаційних системах, заснованих на використанні серверів та баз даних (Клієнт-серверних), виникає проблема зберігання структурованої інформації через мережу Інтернет (формат *JSON*) у клієнтській (фронтенд-частині) програмі. Сервер часто зазнає *CSRF* (*Cross-Site Request Forgery*) та *XSS* (*Cross-Site Scripting*) атак через неправильне зберігання *JSON* веб-ключа. Є декілька варіантів збереження структурованої інформації, всі вони мають свої методи захисту від атак, але все одно залишаються вразливими перед базовими *CSRF* та *XSS* атаками.

Мета статті — показати проблему зберігання структурованої інформації через мережу Інтернет (формат *JSON*) у локальному сховищі (*LocalStorage*) та фрагментів інформації, що передаються у браузер із сайту, який відвідав користувач (файлах cookie), та запропонувати метод збереження *JSON* веб-ключа у локальній змінній всередині замикання (функції, що посилаються на незалежні змінні); на основі авторизації користувача показати взаємодію *JSON* веб-ключа із сервером, та подолання основних проблем авторизації та збереження токену (*JSON Web Token – JWT*).

Методи. Системний підхід, аналіз.

Результати. Здійснено порівняння варіантів збереження *JSON* веб-ключа. Зіставлено приклади наявності токена в зоні ризику *CSRF* та *XSS* атак. Описано алгоритм формування системи безпеки з використанням *JSON* веб-ключа, збереженого в локальній змінній всередині замикання.

Висновки. Результати дослідження показують, що збереження структурованої інформації через мережу Інтернет у локальній змінній всередині замикання (функції, що посилаються на незалежні змінні) має переваги у захисті даних на відміну від збереження токена в локальному сховищі (*LocalStorage*) та фрагментів інформації, що передаються в браузер із сайту, який відвідав користувач. По-перше, система стає вповні захищеною від типових *CSRF* та *XSS* атак. *JSON* веб-ключ у пам'яті не можна отримати під час *XSS* атаки, тому що він не зберігається у локальному сховищі; також наявна захищеність від *CSRF* атак, оскільки кожен *JSON* веб-ключ не може бути відісланий автоматично із фрагментами інформації, що передаються в браузер із сайту, який відвідав користувач, бо токен зберігається в пам'яті, а його відсилання відбувається як заголовок у кожному запиті до сервера. Система захисту при збереженні *JSON* веб-ключа у локальній змінній має гнучкішу архітектуру завдяки покращеному контролю над сутністю *JSON* веб-ключа.

Ключові слова: *JWT, збереження токена, локальна змінна, Cookie, LocalStorage, CSRF атака, XSS атака.*

А.С. Булгакова, кандидат технических наук, доцент,
Николаевский национальный университет имени В.А. Сухомлинского,
54000, Николаев, ул. Никольская, 24, Украина,
sashabulgakova2@gmail.com

В.В. Зосимов, доктор технических наук, доцент, заведующий кафедры информационных технологий,
Николаевский национальный университет имени В.А. Сухомлинского,
54000, Николаев, ул. Никольская, 24, Украина,
zosimovvv@gmail.com

П.Д. Поправкин, магистрант,
Николаевский национальный университет имени В.А. Сухомлинского,
54000, Николаев, ул. Никольская, 24, Украина,
pavel.popravkin.dm@gmail.com

ХРАНЕНИЕ JWT ТОКЕНА В ЛОКАЛЬНОЙ ПЕРЕМЕННОЙ

Введение. При работе с аутентификацией в информационных системах, основанных на использовании серверов и баз данных (Клиент-серверных) возникает проблема хранения структурированной информации через сеть Интернет (формат *JSON*) в клиентской (фронтенд-части) программе. Сервер часто подвергается *CSRF* (*Cross-Site Request Forgery*) и *XSS* (*Cross-Site Scripting*) атакам из-за неправильного хранения *JSON* веб ключа. Существует несколько вариантов сохранения структурированной информации, все они имеют свои методы защиты от атак но все равно остаются уязвимыми перед базовыми *CSRF* и *XSS* атаками.

Цель статьи — показать проблему хранения структурированной информации через сеть Интернет (формат *JSON*) в локальном хранилище и фрагментов информации (*LocalStorage*), передаваемых в браузер с сайта, который посетил пользователь (файлах *cookie*) и предложить метод сохранения *JSON* веб ключа в локальной переменной внутри замыкания (функции, ссылающиеся на независимые переменные; на основе авторизации пользователя показать взаимодействие *JSON* веб ключа с сервером и решение основных проблем авторизации и сохранения токена (*JSON Web Token* — *JWT*).

Методы. Системный подход, анализ.

Результаты. Проведено сравнение вариантов хранения *JSON* веб ключа и опасность *CSRF* и *XSS* атак на представленные методы. Описан алгоритм формирования системы безопасности с использованием *JSON* веб ключа, сохраненном в локальной переменной внутри замыкания.

Выводы. Результаты исследования показывают, что сохранение *JSON* веб ключа в локальной переменной внутри замыкания (функции, ссылающиеся на независимые переменные) имеет более высокую степень защиты данных в отличии от хранения *JSON* веб ключа в локальном хранилище (*LocalStorage*) и фрагментов информации, передаваемых в браузер с сайта, который посетил пользователь (файлах *cookie*) . Система становится полностью защищенной от типичных *CSRF* и *XSS* атак. *JSON* веб ключ в памяти не может быть получен при *XSS* атаке, потому что он не сохраняется в локальном хранилище, также существует защищенность от *CSRF* атак, потому что каждый *JSON* веб ключ не может быть отправлен автоматически с фрагментами информации, передаваемыми в браузер с сайта который посетил пользователь, потому что токен сохраняется в памяти и отправка происходит как заголовок в каждом запросе к серверу. Система защиты при сохранении *JSON* веб ключа в локальной переменной имеет более гибкую архитектуру благодаря улучшению контроля над сущностью *JSON* веб ключа.

Ключевые слова: *JWT, сохранение токена, локальная переменная, Cookie, LocalStorage, CSRF атака, XSS атака.*