Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/190505

Analysis of Authentication Methods and Secure Web Application Realization With an Integrated Authentication System

Abed Saif Ahmed Alghawli

Department of Computer Science, College of Sciences and Humanities, Prince Sattam bin Abdulaziz University, Aflaj, Saudi Arabia

Received: 12 Jan. 2025, Revised: 6 May 2025, Accepted: 11 May 2025
Published online: 1 Sep. 2025

Abstract: The number of cyberattacks is growing every year, and their main goal is to steal personal and confidential data. In most cases, this happens through hacking or theft of web application user credentials due to vulnerabilities in authentication and authorization methods, which in most cases occur due to incorrectly implemented authentication methods. The use of modern authentication methods and their correct use and configuration in web applications are critical features of secure and resilient web applications. This article analyzes the authentication methods for web applications, their vulnerabilities, and a variety of attacks on them, which lead to high risks in their implementation and further use. A standard web application has been created that is similar to the one created based on the Shopify web application builder with authentication based on the Hypertext Transfer Protocol cookie session. The risks of vulnerabilities and attacks on the created web application were analyzed, and considering its results, advantages and disadvantages of authentication methods; the web application was improved: authentication methods, application settings, and security features. The two most secure authentication methods were selected for the web application: JWT Access/Refresh token with browser fingerprints and OAuth 2.0 standard, based on which the improved web application was implemented. A risk analysis of vulnerabilities and attacks on the improved web application has been carried out, which showed that the risks of vulnerabilities and attacks on it are very low. The correct implementation and configuration of the JWT Access/Refresh token authentication method in combination with browser fingerprints is presented, and an analysis of its use is carried out, which shows that this combination provides reliable prevention of token theft and use from another computer. The author also implements authentication using OAuth 2.0 in combination with browser fingerprints and describes its correct implementation and configuration. When analyzing its use, it turned out that delegating authentication to Facebook or Google services can provide a low level of risk of attacks and vulnerabilities on a web application.

Keywords: authentication, integrated authentication system, OAuth, JWT, token, web application, vulnerabilities.

1 Introduction

These days, the Internet has become one of the main standards of the modern person, permeating almost every aspect of our lives, from social media accounts to bank accounts. At the same time, the number of attacks on users' confidential information, information systems of corporations, organizations, critical infrastructure, etc. is increasing [1–4]. Web applications account for almost 90% of Internet resources, but despite their popularity and the fact that they bring significant benefits to all types of businesses, there are increasing security issues. The reasons for this are an increase in the number of cyberattacks and a decline in the quality of web application development. For example, the company Symantec, in its report Global Internet Security Threat Report (ISTR), indicates that cybercriminals usually use vulnerabilities of web applications running on the server or exploit some vulnerabilities of the operating system running these applications [5]. The analysis of web applications by Symantec for the presence of vulnerabilities showed that from 1000 randomly selected web applications it found 80% of problems in the application code itself. On average, the number of vulnerabilities per web application reaches 22, of which 10 are weakly vulnerable, 7 are moderately vulnerable, 3 are highly vulnerable, and 2 are critical. Thus, a clear pattern of errors in code writing and implementation can be observed. This tendency compromises not only confidential user data but also the reputation of the

[*] Corresponding author e-mail: a.alghauly@psau.edu.sa

company itself. In some cases, security measures may be too complex, which can make it impossible to obtain the necessary information even for authorized users [6, 7]. However, information security tools help to ensure the confidentiality, integrity and availability of information in the conditions of various types of threats and reduce the risks of loss or theft of confidential data. The main vulnerabilities to the confidentiality, integrity and availability of web application information are structured query language (SQL) injection attacks, disclosure of confidential data, authentication breaches, access control violations, cross-site scripting (XSS), cross-site request forgery (CSRF); insufficient logging and monitoring. Web application authentication and authorization breaches have been among the top 10 OWASP vulnerabilities for several years in a row [8]. Authentication is one of the key aspects of web application security and involves verifying the identity of users accessing web applications. Authentication is necessary to verify the identity of a user or system that wants to access services and data. Many researchers have analyzed the security of authentication and authorization methods and their implementation in web applications, but only a few research papers and many security vendor analytical reports indicate that most vulnerabilities exist in web applications due to incorrect implementation of authentication and security methods in web applications.

2 Publications Analysis and Problem Statement

At the moment, it is common to use a Single Sign-On (SSO) solution, i.e. one account for authentication on multiple websites, as noted in [9, 10]. This solution is based on using OAuth, OpenID authentication methods, tokens, etc. which, accordingly, should be protected. SSO systems, in their turn, must also comply with high security and confidentiality requirements. However, a researcher, Schmitz Guido [11], detected serious vulnerabilities in SSO systems, leading to critical attacks on their security and confidentiality.

In the study [12], the authors investigated an SSO solution based on the OpenID Connect protocol. They analyzed its stages of operation, advantages, disadvantages, and vulnerabilities. Based on the analysis of the vulnerabilities found, the authors describe the consequences for privacy, availability, and confidentiality associated with user access to SSO systems. In [13], the authors continued to analyze the vulnerabilities of the OpenID Connect protocol and highlighted the possibilities of attacking it. They demonstrated a MIXup attack on accounts of popular platforms, which led to the theft of user tokens and unauthorized access to their data. However, these studies do not indicate how to improve and customize the security of this protocol. In turn, the authors of [14] analyzed the security of using tokens and OAuth, and they conducted SSO testing between systems, token validation analysis, JSON Web Token (JWT) structure verification, and Network Sniffing Attack. However, SSO systems are not only vulnerable to this attack.

The authors of [15] became interested in the security of the OAuth 2.0 protocol used in SSO, and they identified protocol vulnerabilities in case of improper site configuration by testing 75 websites. The authors also developed a browser extension that successfully identifies and warns the user about improper OAuth 2.0 settings. However, the paper does not provide ways to solve security problems on the server side. To provide higher security, the authors of the paper [16] simulated the OAuth 2.0 protocol and suggested additional features based on token use that can improve the architectural design and improve the overall security effectiveness of the protocol but did not propose any specific solutions. The authors of the paper [17] propose their unique approach based on OAuth and token technologies to extend the period of access to secure resources for authenticated users. But it can only solve a narrowly focused issue.

In their research [18] the authors developed a simplified service for user authentication, authorization and management in web and mobile applications based on OAuth and OpenID. However, the developed service is insecure, especially for applications that store vulnerable data. This issue was analyzed by the authors of [19]. The authors propose a method for revoking a token by sending a revocation request to the authorization server when the resource server performs abnormal behaviour using the token, such as logging out or changing the identity of the resource owner. However, the authors do not provide solutions to other security issues.
The research [10] analyses software that supports the SSO technology based on Security Assertion Markup Language (SAML) or OpenID, provides recommendations for a specific choice but again does not provide any solutions to security issues. However, in [20], the authors found problems with the implementation of OpenID components and implemented two new attacks and compromised 12 of the 17 most popular existing OpenID implementations. The authors of [21] showed vulnerabilities in the implementation of 11 SAML frameworks (out of 14 main ones). And the authors of [22] analyzed the OAuth 2.0 protocol and concluded that for most developers who do not have deep knowledge of web security, the implementation of this protocol can be dangerous. The authors of [23] analyzed the OpenID Connect protocol and its specification, found a variety of attacks to which it is vulnerable, and provided countermeasures. However, the authors note that web application developers often do not follow the specification's recommendation for a secure protocol implementation, which leads to user authentication breaches. The authors assume that flaws in protocol implementation will always exist, so they propose a

system that automatically performs and evaluates attacks on web applications against authentication implementation.

Thus, the problem of ensuring secure authentication of web application users and correct implementation is relevant. From the literature review, it is clear that it is necessary to analyze authentication methods and standards for web applications, based on the results of the analysis, to develop a web application with a correctly implemented secure authentication system and to check its quality of implementation.

The purpose of this paper is to analyze authentication methods and secure implementation of authentication methods with the integration of JWT tokens and the OAuth v2.0 standard based on the developed web applications.

3 Review of Authentication Methods

The process of user authentication is preceded by the process of user identification [24]. During the identification process, information about the user is provided in the form of an identifier for the security system. The security system will search for all abstract objects in the set of identifiers and will find a particular object used by the real user. Once it is done, the user is identified. For example, the object can be a user account id. The fact that the user has been identified does not necessarily mean that they are genuine. The user must provide proof of his or her identity in the system, such as a password, which is called a credential in the system. The process of verifying credentials is the process of authentication. If the authentication process is successful, then the user is authorized to the system and granted the access rights assigned to him or her. Authentication is required for a secure web application. Currently, the following authentication methods are the most commonly used [25]:

-cookie-based authentication;
-token-based authentication;
-third-party access (OAuth, Application Programming Interface (API) token);
-OpenID;
-SAML.

3.1 Authentication Based on Hypertext Transfer Protocol Cookie Session

Hypertext Transfer Protocol (HTTP) cookie session is a step towards more reliable and complicated authentication methods. All HTTP requests are stateless [25], i.e. it is impossible to store data about interaction between a server and a user. To solve this issue, a feature of HTTP session was created, allowing web servers to store data (requests/answers) about interaction between a server and a user. They store data based on a specific session (session ID, time of creation, last access, etc.), as well as user data (login status and other data that the application may need from the user). Sessions can be realized by means of cookies, i.e. a small piece of data that the server sends to the user's web browser. The browser can store it and send it back with requests to the same server. Cookies are used for user authentication, contain private information and are small in size, but with a large number of sessions and users, the amount of data to store becomes an issue. Table (1) shows the advantages and disadvantages of authentication based on HTTP cookie sessions.

Table 1: Advantages and disadvantages of authentication based on HTTP cookie sessions.

Advantages	**Disadvantages**
Cookies take up little space.	Cookies are vulnerable to XSS and CSRF attacks.
Cookies are easy to use and apply.	Scaling becomes an issue when many users log in.
Cookies are stored on a server and it is much harder to steal them or replace.	Contain confidential information about the user, which makes them a target for attackers.
Information stored in cookies is encrypted before it is sent, and cookies themselves are transmitted via the HTTPS protocol.	Settings for using and sending cookies depend on the website developer who can make incorrect settings.

3.2 Major Token-Based Authentication Principles

As with the previous cookie session authentication method, there is no specific pattern for this strategy. As a result, all implementations are specific for certain systems. Token-based authentication is most used when developing distributed SSO systems, where one application (a service provider or a relying party) delegates user authentication to another application (an identity provider or authentication service) [25]. A typical example of this method is logging in to the application through a social media account. In this case, social media is an authentication service, and the application entrusts the user authentication function to the selected social network.

This method is as follows - an identification data provider gives reliable information about the user in the form of a token and the service provider application uses this token for user identification, authentication and authorization. The identification data provider is most often used in standards of the OAuth 2.0 type that uses authentication delegating. The identification data provider can be bypassed in some cases of token-based authentication. For example, some implementations with

Table 2: Advantages and disadvantages of using a JWT token.

Advantages	Disadvantages
Server scalability does not change with an increase in the number of users.	JWT is much bigger than a cookie session identifier.
JWT is stateless - a web application is not required to store session data on a server, which reduces the server load, improves performance and scalability.	JWT cannot revoke access to the user.
JWT contains more information about the user.	A token is stored on the user's side, which makes it vulnerable for attackers.
Portability - enables cross-domain and cross-platform authentication and authorization.	Vulnerability to theft, i.e. if someone steals a JWT from the user or system, they can use it to misappropriate their identity and get access to their resources.
Adaptability - includes any information relating to a web application that allows access to be controlled in a precise and personalized manner.	

a JWT token do not require an additional identification data provider to generate the token. The server itself creates the token and sends and receives it from the client.

3.3 JSON Web Token-Based (JWT) Authentication

JWT is a mechanism used for checking the owner of some JSON data. It is an encoded line which can contain an unlimited amount of data (unlike a cookie) and has a cryptographic signature [26]. When a server gets a JWT, it can guarantee that data in this line is reliable as it is signed with a cryptographic signature. No intermediary can change a JWT once it has been sent. It should be noted that a JWT guarantees data ownership but not encryption, i.e. JSON data stored in a JWT is not encrypted and it can be seen when a token is intercepted. In this connection it is highly recommended to use HTTPS with a JWT. One of the most complicated issues about a JWT is where to store a token. Thus, JWT-based authentication security will depend on the settings of the website developer. A token must be stored in a secure place in the user's browser: a local storage/session storage, a cookie, or application memory.

In the first case, if a token is stored inside a Local Storage or Session Storage, it is available to every script on the web page. A Cross-site scripting (XSS) attack can allow an attacker to gain access to tokens. It is not recommended to store tokens in a local storage or session storage. In the second case, tokens stored in cookies are vulnerable to CSRF (Cross-site request forgery) attacks. Therefore, it is also not recommended to store them in cookies.
If any of the scripts included in the web page are discredited, the attacker will be able to access all the tokens stored in browsers. That is why the best solution is to store the token in the application memory itself, which cannot be accessed by conventional methods. Table (2) shows the advantages and disadvantages of using a JWT token.

3.4 Access/Refresh Token-Based Authentication

To make JWT-based authentication more secure tokens are divided into Access and Refresh types [27, 28]. An access token is used to authorise requests and store additional information about the user. A refresh token is given by a server upon successful authentication and is used to obtain a new pair of Access/Refresh tokens. It is most commonly stored in a server database. Each token has its own lifespan, for example, Access - 30 minutes, while Refresh expires after 60 days. The Refresh token is stored on the server to keep track of access and disable stolen tokens. This way, the server determines exactly who is allowed to log in.

To make authentication possible on more than one device, you need to store all Refresh tokens for each user. At the moment of Refresh, i.e. when the Access token is updated, both Access and Refresh tokens are updated. At the moment of Refresh, the Refresh token compares itself with the Refresh token in the database, and if they match and the refresh token has not expired yet, the system updates the tokens. The Refresh token has a lifespan in case the user is offline for more than 60 days, in which case they will have to re-enter their login/password.

A browser fingerprint is a tool for identifying a user's browser. It is hash generated on the basis of some unique browser parameters. The advantage of a fingerprint is that when it is generated, this value is unique for a particular user's browser and will not change in the future. Therefore, it is very difficult to compromise a browser fingerprint.

3.5 SAML Authentication

SAML is a standard of authentication and authorization, a variant of an extensible markup language XML for information exchange about security on the Internet.
SAML exchange is done between system entities referred to as a SAML asserting party (also called SAML authority) and a relying party that processes assertions it has received [25, 27]. Security assertion is a standardized assertion in a markup language that takes solutions regarding access control.

Table 3: Advantages and disadvantages of the most common authentication methods.

Method	Advantages	Disadvantages
Cookies	Easy to implement and widely supported, it stores data about a session, settings or other data necessary for user identification or customizing user experience.	Vulnerable to CSRF attacks, add overheads to each request, limit the size and number.
JWT	Stateless, can carry more data, can support several domains or services, and can be checked by anyone who has access to the key.	Vulnerable to XSS attacks, has a fixed lifespan, and cannot be easily withdrawn or updated.
OAuth	Allows the user to provide access to their resources or data from one website to another website without exchanging credentials and can use different types of tokens, such as JWT, API tokens or SAML assertions.	Complicated and requires the participation of many parties and interactions, it creates definite security risks, such as phishing attacks, token leakage or repeated playback.
SAML	Allows the user to log in to one website and access another website without re-entering credentials, uses assertions that contain user identity data, attributes or authorization decisions, can be signed and encrypted by the identity provider and verified by the service provider.	It is complicated and requires XML processing and parsing, and has some performance issues due to the size and number of messages involved.
OpenID	Allows the user to log in to one website and use their identifier to access another website without creating an account or re-entering credentials, can use different types of tokens to represent identification information.	Complicated, requires the participation of many parties and interaction, and creates definite security risks, such as phishing attacks, token leakage or forgery of identifiers.

SAML-based single sign-on (SSO) can be categorized into two primary approaches: identity provider (IdP)-initiated and service provider (SP)-initiated. In both cases, the IdP handles user authentication, but they differ in their initiation processes. In IdP-initiated SSO, the user begins by logging into the IdP, which then directly authenticates the user and issues a SAML response. Conversely, SP-initiated SSO starts when the user attempts to access the SP. Here, the SP generates a SAML request, redirects the user to the IdP for authentication, and, upon successful verification, returns the user to the SP to finalize the login process.

3.6 OpenID Connect Authentication

OpenID Connect is an authentication protocol built on OAuth 2.0, enabling standardized user authentication and secure identity data sharing with third-party applications [25, 27]. The key distinction between OAuth 2.0 and OpenID Connect lies in their token systems. OAuth 2.0 primarily issues short-lived access tokens, which grant applications permission to access specific resources. In contrast, OpenID Connect generates identity tokens, which verify user identity and transmit authenticated user data to applications. While both protocols employ similar operational workflows, their token purposes differ fundamentally: access tokens facilitate authorization (resource access), whereas identity tokens handle authentication (identity verification).

3.7 Advantages and Disadvantages of Different Authentication Methods

Several methods are used to authenticate web applications, namely cookies, JWT, OAuth, API Token, SAML and OpenID described above. Depending on the usage scenario and security requirements, they have different advantages and disadvantages, as shown in table (3).

Since it is difficult to draw conclusions from the general comparison, we will make a more detailed comparison of Cookie and JWT token, OAuth and SAML authentication methods.

4 Comparison of Authentication Methods

4.1 Comparison of Cookie and JWT Token Application

At the moment the most widely used authentication methods are Cookie sessions and a JWT token [27]. Table (4) shows the results of the comparison of these 2 methods by such parameters as scalability, security, RESTful API services, and performance. It should be noted that a standard implementation of a JWT token algorithm without any specific changes is considered. An example is a Refresh token.

Taking into account the results of the analysis it can be concluded that a cookie session authentication method fades into the background compared to a JWT token. In the context of using a JWT token with small-size applications, a usual JWT authentication without a refresh token can

Table 4: Parameters of using Cookie and JWT token authentication methods.

Parameters	**Cookie sessions**	JWT token
Scalability	Session data is stored in memory on a server or in a database. In the horizontal scaling scenario, a separate central session storage system is created that can be accessed by all application servers.	Easy to scale, as tokens can be used to access resources from different servers, which saves costs, as no separate server is required to store user sessions.
Security	In the case of cookie sessions, all data is stored on a server, so it is considered relatively secure.	Due to signing with a secret key on the server, a token cannot be changed undetected without access to the secret key. Storing the token anywhere other than in the application memory is very risky.
API RESTful Services	The state of the application is saved. The API interface is often served from one server, but in reality, the application uses it from another.	RESTful API is stateless, thus it is not important where API or an application is serviced.
Performance	Overheads are not high because when encoding the size of JWT will be several times the size of the SESSION identifier.	A significant load for each HTTP request. However, JWTs trade size for the ability to store data on the client side. For example, privileges in the token itself or user id.

be used, which will completely replace cookie sessions. If it is necessary to use a JWT with middle and large-size applications, a combination of access/refresh tokens and browser fingerprints can be used.

4.2 Comparison of OAuth and SAML Authentication and Authorization Methods

To achieve the objectives of the paper and perform authentication in a web application, it is necessary to use a single sign-on (SSO) account. The advantage of the SSO account is that users log in once using a single set of credentials, and they can get access to multiple services and applications during a single session [27]. SSO is used to control authentication and secure access to corporate networks, web applications, etc. The most commonly used standards for SSO implementation are SAML (Security Assertion Markup Language), OAuth and, less commonly, OpenID Connect. The results of the analysis of the use of SAML and OAuth methods for the implementation of single sign-on are shown in table (5). Taking into account these comparisons, we have chosen 2 authentication methods to be implemented in a web application: JWT tokens and OAuth 2.0. So let us consider web application protection technologies used for implementing a secure web application by means of these authentication approaches.

5 Description of Technologies Used for Web Application Implementation

We will analyze the authentication security of a web application represented by a standard online store selling electrical goods, which was developed and is standard when ordering its development and is analogous to those developed using the Shopify web application builder [29]. A web application is a division of a server module and a client module. This division and technologies used in the development allow us to call the client side of the application a Single page application. A Single page application (SPA) is a web application that interacts with a web browser dynamically rewriting a current web page with new data that was obtained by sending a request to the web server endpoint, instead of the browser loading entire new pages by default [27, 30]. The web application implements the basic standard functionality of a modern web application: Hypertext Transfer Protocol cookie session-based authentication, access control with a division into client and administrator user types, a mechanism for working with the SQL database, and a mechanism for working with the PayPal service. The libraries used to develop the client side of the application are shown in table (6).

A server side of the application was created in the nodejs development environment with the express auxiliary library and libraries shown in table (7), since the server side does not include complex functionality in working with data coming to the server.

Let's conduct a qualitative analysis of attacks and vulnerability risks for the standard proposed web application based on the OWASP methodology and the method of expert assessments [31], the results of which are presented in table (8).

For a quantitative risk assessment the resulting risk is assigned values from 1 to 5, where 1 is a very low risk, 2 is low, 3 is average, 4 is high and 5 is very high. Correspondingly, values of the resulting risk for a standard web application of an online shop is 49 with a maximum risk value of 55 and a minimum risk value of 11. It is clear that standard implementation of the web application has a big number of serious vulnerabilities the risks of which are very high. That is why this web application will be improved and a security system using advanced vulnerability prevention methods and the best

Table 5: Parameters for using Security Assertion Markup Language and Open Authorization authentication methods.

Parameters	**SAML**	**OAuth**
Workflow	Relies on XML-based security assertions exchanged between a confirming party (SAML authority) and a relying party.	Based on the interaction between the resource owner, resource server, client (application) and authorization server.
SSO (Single Sign-On)	Enables Single Sign-On (SSO) by securely transferring authentication and authorization data between web servers.	Facilitates SSO by allowing authenticated users to grant third-party applications access to their data without sharing credentials.
Security	It is considered to be more secure due to the ability to encrypt assertions, which is suitable for processing confidential data.	Uses delegated access tokens (not encrypted assertions), prioritizing streamlined authorization over assertion security.
Interoperability	Preferred by organizations with existing XML infrastructure or those requiring federated identity management.	Supports various types of clients, including web, mobile and desktop applications, making it valuable for developers of mobile applications and services on the open web.
Major Factors	Evolution of SSO, expansion of federated identity management, changing industry standards.	Simplicity for developers, need for limited exchange of user information with other applications, and joint advertising activities between large Internet companies.
Encoding and Security	Supports assertion encryption, ensuring secure data exchange in high-stakes environments.	Focuses on token-based access control without built-in token encryption standards.
Compatibility	This can work alongside OAuth in systems that need both authentication and authorization to manage access control.	May use SAML for initial identity verification before issuing OAuth tokens for resource access.
Usage Options	Multi-domain SSO is ideal for large organizations and enterprise applications such as Salesforce and Marketo	User privacy provides access to private resources on different websites or in different applications without transferring user identification data.
Recommendations for Implementation	Optimal for organizations investing in XML-based security and federated authentication.	Recommended for modern mobile applications, as well as in cases where API access is a priority, especially on the open Internet.

Table 6: List of technologies used on the client side.

Name	**Description of the technology**
React	Library used as a basis for creating SPA applications
MobX	A library that allows the client side to store data in a centralized storage. This storage simplifies the application development.
MobX-router	A library that connects to the MobX centralized storage and provides an easy way to navigate the application
MobX-react	A library that makes it possible to use react and MobX in combination
antd	A library that provides a large number of ready-made solutions for implementing the visual part of the application
axios	A library to make it easier to work with network requests
webpack	A library to facilitate general development and efficiently collect the code of the client side to send it to the hosting

Table 7: List of technologies used on the server side.

Name	**Description of the technology**
express	Library used as a basis for building the server side
body-parser	The library is needed to process and understand the data coming to the server
axios	Library to facilitate work with network requests
http-error	Library for standardized generation of http errors
moment	Library for standardized work with dates
lodash	A library that provides a large number of useful functions for working with different types of data
dotenv	Auxiliary library for processing files with the .env extension
PayPal-node	A library that interacts with the PayPal money transaction service

Table 8: Qualitative risk assessment for a standard web application of an online shop.

Name of a Risk	Vulnerability Type	Losses	Probability of Occurrence	Resulting Risk
Sending malicious code to a server from a client through unsecured input fields	SQL injection	Very high	High	High
Receiving and processing malicious code by a server	SQL injection	Very high	Very high	Very high
Password cracking	Authentication violation	High	Very high	Very high
Session token theft	Authentication violation	High	Very high	Very high
Disclosure of passwords in case of unauthorised access to the database	Disclosure of confidential information	Very high	Very high	Very high
Confidential data theft during traffic interception	Disclosure of confidential information	Very high	Very high	Very high
Gaining unauthorised access to an administrator account	Violation of access control	High	High	High
Gaining unauthorised access to administrator functions	Violation of access control	High	High	High
Entering malicious code into the client side of an application	XSS	High	Low	Average
Vulnerability of sending confidential data via an attacker's website	CSRF	High	Average	High
Late detection of vulnerabilities in the system	Insufficient logging and monitoring	Very high	Very high	Very high

authentication methods will be built. Such a system will minimise the risk of attacks and reduce the probability of vulnerabilities being exploited.

5.1 Technologies of the Server Side of the Improved Web Application

While improving the web application and implementing secure authentication methods it is necessary to develop a web application server having chosen secure, fast and effective technologies. The server side is based on the Nodejs technology, which is used with the Express framework, as in the standard implementation of the web application. For regular authentication, we used the jsonwebtoken library, which allows us to generate and verify a JWT. Passportjs authentication library together with passport-facebook-token, a complimentary authentication strategy, were used as OAuth 2.0 authentication. Hosting and certification of the server is provided by the Heroku cloud service. PostgreSQL, also hosted by Heroku, is chosen as a database. A more detailed list of the libraries used in the development of the server side of the application is given in table (9).

Now it is necessary to choose technologies to be used on the client side.

Table 9: A detailed list of the libraries used on the server.

Name	**Library description**
bcryptjs	Library for encoding data on the basis of a secret
body-parser	This library is needed for the server to read data coming from the client side of the web application
cookie-parser	This library is needed for the server to read cookies coming from the client side of the web application
cors	This library is needed for the server to perceive requests from the client side of the web application
pg	This library is needed for interaction with PostgreSQL database
express-promise-router	Library that helps the server to define endpoints which the client will access
express-dynamic-middleware	Complimentary library for authentication

5.2 Technologies of the Client Side of the Improved Web Application

JavaScript used with the React framework was taken as a base of the client part. To initiate OAuth 2.0 authentication, we used the react-facebook-login library, which provides an opportunity to start the authentication process using the Facebook service. Hosting and

Table 10: A detailed list of the libraries used on the server

Name	Library description
React-router	The library that helps to navigate between pages on the client side and ensures safe routing.
Axios	The library that helps the client side to make requests to the server and add all the necessary information to these requests.
Redux	Library that provides the client side with a centralized data repository. A number of other supporting technologies follow from this technology.
Redux-thunk	The library that helps the client-side work with server requests.
Redux-form	The library that helps the client side to work with forms and check the validity of data submitted to this form.
material-ui	The library for facilitating work with the visual part of the client side.
Js-cookie	The library for working with cookies on the client side.

certification of the client is provided by the Heroku cloud service. A more detailed list of the libraries used in the development of the client side of the web application is shown in table (10).

Authentication of the application will be made in 2 ways:

–JWT token authentication for users who don't use Facebook;
–OAuth 2.0 authentication and Facebook authentication.

JWT Token Authentication

To complete JWT Token Authentication in the application we used a strategy with the use of an Access / Refresh token. To identify a user, browser fingerprints are applied, which allows you to track the second browser from which the request to the server was made.

After sending user data (email, login, password, browser fingerprint) to the server, it registers the user and adds this data to a PostgreSQL database. All passwords in a PostgreSQL database are stored in a salted hashed format for this application implementation. It is done to prevent user password theft if a database is compromised by an attacker. After registering the user can reenter their data and authenticate for further authorization on the website.

Website authentication is done in such a way that the server compares user data. If the user is found in a database, then they are given Access/Refresh tokns that will be used for access to secure API endpoints. To prevent new Access tokens from being tampered with, a Refresh token additionally compares browser fingerprints and, if the data matches, issues a new Access token. Otherwise, the Refresh token is destroyed. This destruction initiates re-authentication.

OAuth 2.0 Authentication.

In addition to conventional authentication, the application also implements OAuth 2.0 authentication. Figure 1 shows a schematic view of OAuth 2.0 authentication in this application.

In this implementation an authentication strategy allows the user to log in to the website from Facebook. The client who authenticates through Facebook entrusts access to some of their data. The server, in its turn, gains access to Facebook with the user's data and gives the user who has been authenticated through Facebook the full right to access and use protected information from secure API endpoints.

The authentication process begins on the client side when the client provides his or her Facebook account details via OAuth. It is done with the help of a pop-up window presented by the Facebook service. Data is transmitted in this window using the HTTPS protocol, so the possibility of data theft during sending is minimal. After granting permission, the client receives a token that will be sent to the server. Once sent, the server processes the token and transmits it back to Facebook in exchange for the data of the client who sent the token.

The list of data that comes from Facebook can be customized by the user. After the user confirms the list of data, the server saves the new user's data (browser fingerprints, Facebook refresh token, Facebook ID) to a separate collection in the database, which is designed specifically for those who have authenticated via Facebook. After saving the user into the database, the server sends the user Refresh and Access tokens generated by Facebook and the user's profile data from Facebook. The next time the user logs in, the server will be able to request this data from the database and verify the user's authenticity.

6 Efficiency Analysis of the Proposed Authentication Methods

To define the quality of the implemented web application authentication methods, a qualitative assessment was carried out during authentication with the help of the JWT Access/Refresh token and OAuth 2.0 using browser fingerprints. The web application and security features were configured as described above. Attacks and vulnerabilities risks for the developed web application were assessed on the basis of the OWASP methodology and the expert assessments method [31] applying JWT Refresh and Access tokens and OAuth 2.0 authentication

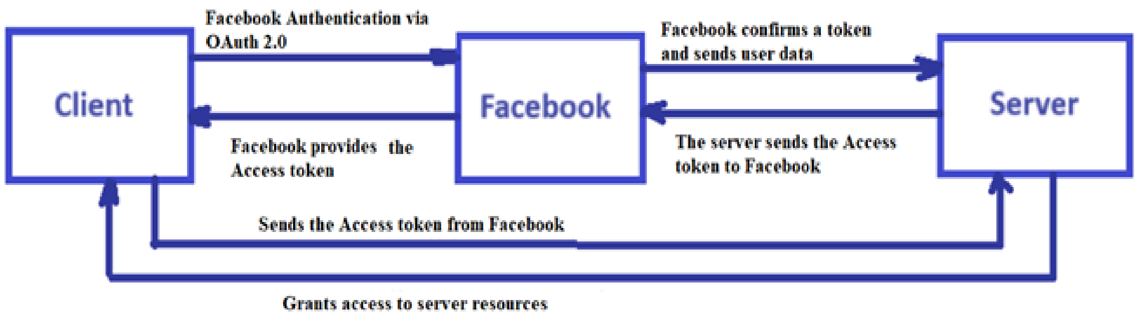
Client
Facebook Authentication via OAuth 2.0
Facebook provides the Access token
Facebook
Facebook confirms a token and sends user data
The server sends the Access token to Facebook
Server
Sends the Access token from Facebook
Grants access to server resources

Fig. 1: OAuth 2.0 scheme in the application implementation

Table 11: Qualitative risk assessment of a web application with the implementation of security functions

Name of a Risk	Vulnerability Type	Losses	Probability of Occurrence		Resulting Risk	
			JWT	OAuth	JWT	OAuth
1	2	3	4	5	6	7
Sending malicious code to a server from a client through unsecured input fields	SQL injection	Low Very	low	Very low	Very low	Very low
Receiving and processing malicious code by a server	SQL injection	Low Very	low Very	low	Very low	Very low
Password cracking	Authentication violation	Low Very	low	Very low	Very low	Very low
Session token theft	Authentication violation	Average	Average	Low	Average	Low
Disclosure of passwords in case of unauthorised access to the database	Disclosure of confidential information	Low	Very low	Very low	Very low	Very low
Confidential data theft during traffic interception	Disclosure of confidential information	Very low	Very low	Very low	Very low	Very low
Gaining unauthorised access to an administrator account	Violation of access control	Average	Very low	Low	Low	Low
Entering malicious code into the client side of an application	XSS	Very low	Very low	Very low	Very low	Very low
Vulnerability of sending confidential data via an attacker's website	CSRF	Low	Very low	Very low	Very low	Very low
Late detection of vulnerabilities in the system	Insufficient logging and monitoring	Low	Average	Average	Average	Average

approaches. The results of the assessment are shown in table (11).

For a quantitative risk assessment, the resulting risk is assigned values from 1 to 5, where 1 is a very low risk, 2 is low, 3 is average, 4 is high, and 5 is very high. The maximum risk value is 55, and the minimum risk value is 11. Correspondingly, the value of the resulting risk for JWT Refresh and Access token authentication is 15, and for OAuth 2.0 is 14, which is a rather low value. As can be seen from Table 11 two points have an average risk level, namely "Late detection of vulnerabilities in the system" and "Session token theft". As for late detection of vulnerabilities, it depends on the human factor. No matter how reliable a logging system is, a person may still not notice a threat that has already been logged. Regarding the session token theft vulnerability: since the role of session tokens is played by JWT cookies, they can still be stolen through the Stealer malware. In this case, theft can be carried out directly from the victim's file system when Stealer malware enters the victim's computer, which depends on its security. Thus, using OAuth 2.0 with browser fingerprints is more secure than

JWT Refresh and Access token authentication, but the difference in risk is minimal, so any of the implemented protocols can be used.

7 Conclusion

In this paper, the tasks of analyzing authentication methods and securely implementing authentication methods with the integration of JWT tokens and the OAuth v2.0 standard based on developed web applications were solved. For this purpose, the most known authentication methods have been analyzed: Hypertext Transfer Protocol cookie sessions, JWT tokens, OAuth 2.0, OpenID, and SAML. It has been established that each of these authentication methods has disadvantages, which means that there are risks when using them and the need to configure these methods in their software implementation. A standard web application has been created for an online shop with the help of the Shopify website builder with authentication based on the Hypertext Transfer Protocol cookie session. The vulnerabilities and attack risks of this web application have been analyzed. The analysis shows that vulnerabilities and attack risks on the application are very high, which implies the need to improve the authentication, settings and security features of the web application. Taking into account the analysis of vulnerabilities and attack risks of the developed standard web application and the advantages and disadvantages of authentication methods, the web application has been improved: authentication methods, application settings and security features. For authentication, we chose to use a combination of a JWT Access/Refresh token with browser fingerprints and OAuth 2.0 delegating authentication to the Facebook service. An analysis of vulnerabilities and attack risks for this web application has been carried out, which showed that these risks for the improved application are very low. It is shown that one of the most secure methods of ensuring user data security in a web application is the use of a combination of a JWT Access/Refresh token and browser fingerprints. Another secure authentication method implemented in the web application is the OAuth 2.0 authentication method with delegation of authentication to the Facebook service. Configuration of this method has shown that vulnerabilities and attack risks for the web application are very low when using it. It is noted that such authentication methods can be compromised only when the client sends initial information with browser fingerprints for the first time, but the probability of this is very low since all information is always transmitted via HTTPS secure connection. The results of this research should be used to implement authentication in web applications from small to medium sizes. In the future, it is necessary to improve the level of protection of the web application and the authentication process, as well as approaches to assessing the quality of the developed web application.

Acknowledgments

This study is supported via funding from Prince Sattam bin Abdulaziz University [project number: PSAU/2025/R/1446].

Conflicts of Interest

The author declares no conflict of interest.

References

[1] B. W. Loo, P. L. Tan and W. Y. Tey, S. K.and Chin, Authentication methods selection in information security through hybrid ahp and egt, *Journal of Advanced Research in Applied Sciences and Engineering Technology* **50**(2) (2025) 171–185.

[2] T. Radivilova, I. Dobrynin, O. Lemeshko, D. Ageyev and A. Ilkov, Analysis of approaches of monitoring, intrusion detection and identification of network attacks, *IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S & T), Kharkiv, Ukraine* (2021) 631–634.

[3] T. Radivilova, L. Kirichenko, A. S. Alghawli, A. Ilkov, M. Tawalbeh and P. Zinchenko, The complex method of intrusion detection based on anomaly detection and misuse detection, *IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Kyiv, Ukraine* (2020) 133–137.

[4] M. TajDini, V. Sokolov, I. Kuzminykh and B. Ghita, Brainwave-based authentication using features fusion, *Computers & Security* **129** (2023) p. 103198.

[5] Symantec's annual threat report reveals more ambitious and destructive attacks. 19 Feb, 2019 https://symantec-enterprise-blogs.security.com/threat-intelligence/istr-24-cyber-security-threat-landscape.

[6] D. Holubnychyi, V. Martovytskyi, I. Ruban, O. Sievierinov, V. Lebediev and V. Tretiak, Functional model of computer networks security information, in *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T)*, IEEE2021, pp. 559–563.

[7] Z. Hu, S. Petoukhov, I. Dychka and M. He, Advances in computer science for engineering and education ii, in *International Conference on Computer Science, Engineering and Education Applications (ICCSEEA)*, Springer Cham2020.

[8] Owasp top 10 api security risks – 2023 https://owasp.org/API-Security/editions/2023/en/0x11-t10/.

[9] A. C. and C. M., The prevalence of single sign-on on the web: Towards the next generation of web content measurement, in *Proceedings of the 2023 ACM on Internet Measurement Conference*, DOI:10.1145/3618257.3624841.

[10] N. Shaikh, K. Kasat and S. Jadhav, Secured authentication by single sign on (sso): A big picture, in *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, IEEE2022, pp. 951–955.

[11] G. Schmitz, Privacy-preserving web single sign-on: Formal security analysis and design, *It-Information Technology* **64**(1-2) (2022) 43–48.

[12] M. Al-Shabi, A. Bennour, M. Al-Sarem, O. I. Khalaf, R. Sikder and S. Algburi, Enhancing security in openid connect single sign-on (sso) systems: a comprehensive vulnerability analysis and mitigation approach, in *4th International Conference on Distributed Sensing and Intelligent Systems (ICDSIS 2023)*, **2023**, IET2023, pp. 50–60.

[13] M. Al Shabi and R. R. Marie, Analyzing privacy implications and security vulnerabilities in single sign-on systems: A case study on openid connect., *International Journal of Advanced Computer Science & Applications* **15**(4) (2024).

[14] E. S. Mansur, A. Rahmatulloh, R. N. Shofa and I. Darmawan, Aman: Token-based authentication to improved single sign-on security between systems, in *2023 International Conference on Advancement in Data Science, E-learning and Information System (ICADEIS)*, IEEE2023, pp. 1–6.

[15] S. Sharma and K. Jevitha, Security analysis of oauth 2.0 implementation, in *2023 Innovations in Power and Advanced Computing Technologies (i-PACT)*, IEEE2023, pp. 1–8.

[16] J. Singh and N. K. Chaudhary, Oauth 2.0: Architectural design augmentation for mitigation of common security vulnerabilities, *Journal of Information Security and Applications* **65** (2022) p. 103091.

[17] F. Al-Husari, Designating a leader browser tab to perform refreshing of access token in oauth 2.0, in *2023 International Scientific Conference on Computer Science (COMSCI)*, IEEE2023, pp. 1–4.

[18] D. Shevchuk, O. Harasymchuk, A. Partyka and N. Korshun, Designing secured services for authentication, authorization, and accounting of users, *Cybersecurity Providing in Information and Telecommunication Systems* **3550** (2023) 207–225.

[19] J. Park, J. Kim, M. Park and S. Jung, A study of oauth 2.0 risk notification and token revocation from resource server, in *Information Security Applications: 16th International Workshop, WISA 2015, Jeju Island, Korea, August 20–22, 2015, Revised Selected Papers 16*, Springer2016, pp. 281–287.

[20] C. Mainka, V. Mladenov and J. Schwenk, Do not trust me: Using malicious idps for analyzing and attacking single sign-on, in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE2016, pp. 321–336.

[21] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann and M. Jensen, On breaking {SAML}: Be whoever you want to be, in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 397–412.

[22] S.-T. Sun and K. Beznosov, The devil is in the (implementation) details: an empirical analysis of oauth sso systems, in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 378–390.

[23] C. Mainka, V. Mladenov, J. Schwenk and T. Wich, Sok: single sign-on security—an evaluation of openid connect, in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE2017, pp. 251–266.

[24] P. A. Grassi, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. Greene *et al.*, Digital identity guidelines: authentication and lifecycle management (2017).

[25] S. K. Dash and S. Dash, *Ultimate Web Authentication Handbook*

[26] M. Jones, J. Bradley and N. Sakimura, Json web token (jwt), tech. rep. (2015).

[27] I. Alsmadi, R. Burdwell, A. Aleroud, A. Wahbeh, M. A. Al-Qudah and A. Al-Omari, Practical information security, *Cham: Springer* **78**(3) (2018).

[28] Rfc 6749. oauth 2.0 refresh token. 2020 https://oauth.net/2/grant-types/refresh-token.

[29] 10 best e-commerce website builders compared in 2024. 2024. https://www.websiteplanet.com/blog/best-website-builders-ecommerce-websites/.

[30] Single page apps in depth. 2013 http://singlepageappbook.com/goal.html.

[31] A. S. A. Alghawli and T. Radivilova, Resilient cloud cluster with devsecops security model, automates a data analysis, vulnerability search and risk calculation, *Alexandria Engineering Journal* **107** (2024) 136–149.