

Security methods in .NET technology^{*}

Serhii Buchyk^{1,*†}, Serhii Toliupa^{1†}, Oleksandr Buchyk^{1,†}, and Anastasiia Shabanova^{1,†}

¹ Taras Shevchenko National University of Kyiv, 60 Volodymyrska str., 01033 Kyiv, Ukraine

Abstract

This article is devoted to the analysis of security methods in .NET technology, which are key to ensuring the reliability and security of software. The article discusses in detail such security methods as authentication and authorisation, encryption, access control, protection against Cross-Site Scripting (XSS) attacks and protection against Cross-Site Request Forgery (CSRF) attacks. The article analyses the advantages and disadvantages of each method, emphasising their importance in forming a comprehensive security system for .NET applications. The results of the study confirm that the implementation and use of various security methods is necessary to ensure the highest level of data and resource protection on the .NET platform. This article will be useful for developers and administrators seeking to improve the security of their .NET applications, ensuring their resilience to modern cyber threats.

Keywords

.NET technology, authentication, authorisation, encryption, hashing, XSS, CSRF 1

1. Introduction

Security is a key issue for any IT system in today's world, where more and more information is stored and processed in a digital environment. NET technology, which is widely used to develop web applications and services, is no exception. It is becoming increasingly important for individuals and large organisations to protect data, prevent unauthorised access and prevent cyber-attacks. With the ever-increasing number of cyber threats and malicious attacks, a thorough understanding and implementation of security practices in .NET is becoming a critical task for developers and system administrators. Securing .NET software involves various approaches to authentication and authorisation, data encryption, access control, and mechanisms to protect against common attacks such as XSS and CSRF.

Security techniques in .NET include the use of authentication and authorisation with various approaches such as Windows Authentication, Forms Authentication, Claims-based Authentication, OAuth 2.0, OpenID Connect and Role-Based Authorisation. Encryption also plays an important role in data protection, using both symmetric and asymmetric encryption and hashing. Access control is provided through ASP.NET membership, role-based access control, token-based authentication and secure protocols such as WebSocket. Special attention is paid to protecting against XSS attacks using input validation, output encoding, XSS protection libraries, HttpOnly cookies and content security policy, and protecting against CSRF attacks using anti-CSRF tokens, HTTP headers and SameSite cookies.

Each of these methods has its own advantages and disadvantages, but together they form a comprehensive security system that provides a high level of security for software developed on the .NET platform. In this study, we will look at the main security methods, their features and effectiveness in preventing cyber threats, so that developers can choose the best approaches to ensure the reliability and security of their applications.

^{*} CPITS-II 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2025, Kyiv, Ukraine

^{*} Corresponding author.

[†] These authors contributed equally.

✉ buchyk@knu.ua (S. Buchyk); toliupa@i.ua (S. Toliupa); alex8sbu@knu.ua (O. Buchyk); shabanovaa@knu.ua (A. Shabanova)

ORCID 0000-0003-0892-3494 (S. Buchyk); 0000-0002-1919-9174 (S. Toliupa); 0000-0001-7102-2176 (O. Buchyk); 0009-0008-4962-569X (A. Shabanova)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Task formulation

The purpose of the study is to analyse security methods in .NET technology. The object of research is the process of analysing security methods in .NET technology. The subject of the study is security methods in .NET technology. Therefore, the task is to study and compare the security methods in the .NET technology.

3. Solving the task

.NET is a software development platform developed by Microsoft Corporation that allows developers to create a variety of applications for different devices and operating systems, as well as mobile devices, personal computers and cloud services [1–7]. The platform includes programming languages, runtime environments, and library classes that help developers ensure the functionality of their applications. One of the key components of the .NET technology is the C# programming language, which is one of the most popular programming languages in the world. In addition to C#, the platform supports other programming languages such as F#, Visual Basic, C++/CLI and PowerShell, as well as non-Microsoft languages (e.g. Cobol, Java, PHP, Python, Scheme). There is also support for other programming languages (approximately 25 languages) [8, 9].

The runtime, libraries, and languages are the pillars of the .NET stack. Higher-level components such as .NET tools and application stacks such as ASP.NET Core are built on top of these pillars. These pillars have a symbiotic relationship because they were designed and built together by the same group. In essence, the .NET platform has a different way of packaging and installing. Instead of being built into the operating system, .NET is compiled from NuGet packages and can be compiled directly into an application or placed in a folder within the application. This means that applications can include .NET and co-exist fully on a computer. NuGet is a package management tool for .NET that has a significant number of packages (hundreds of thousands of packages) that implement a variety of functionality for many scenarios. Most applications rely on NuGet packages for some functionality [9].

Security practices in .NET technology play an important role in ensuring the security and protection of software developed on this platform. The methods considered include different approaches to authentication, authorisation, encryption and access control, as well as protection against different types of attacks, such as XSS and CSRF [10]. A generalised system of security methods in .NET technology is shown in Figure 1.

In .NET, authentication and authorisation are achieved through a combination of middleware components, attributes and configuration settings. The framework provides a flexible and extensible infrastructure for implementing these security mechanisms, allowing developers to choose from a variety of authentication schemes and authorisation policies to meet the requirements of their applications [11].

Windows Authentication is an authentication method that uses existing user accounts in the Windows operating system to authenticate to .NET Web applications. This method is convenient for users because they do not have to enter a separate username and password, as the system uses the same credentials they use to log in to the operating system. Windows-based authentication is performed between the Windows server and the client machine, specifically using Active Directory domain identifiers or Windows accounts for identification [12].

Forms authentication uses HTML forms to collect usernames and passwords from users. The data entered is then checked against your application's user database [13]. If the authentication process is successful, the system stores the authentication token in a cookie or in the URL so that the authenticated user does not have to re-enter the credentials for subsequent requests. When a user accesses an ASP.NET page through a browser, ASP.NET checks for the presence of a form authentication token. If the result is positive, the runtime redirects the user to the login page and begins the process of verifying the user ID and password [14].

Claims-based authentication is a method that uses tokens to authenticate users. The token contains a set of claims about the user, such as username, role, etc. This token is issued by the authentication server after a successful login. This data is sent as a stream of bytes during transmission over the network, and the claims are digitally signed for verification at the receiving end. In the claims-based authentication process, a request to verify a user's identity is sent from a Dynamics 365 customer experience or custom application to a Security Token Service (STS) server. The STS server determines whether the user requires authentication and, if so, issues a signed and encrypted Security Assertion Markup Language (SAML) token containing the user's authentication credentials. The token has a limited validity period and may need to be updated periodically, depending on the length of time it is used by the application [15].

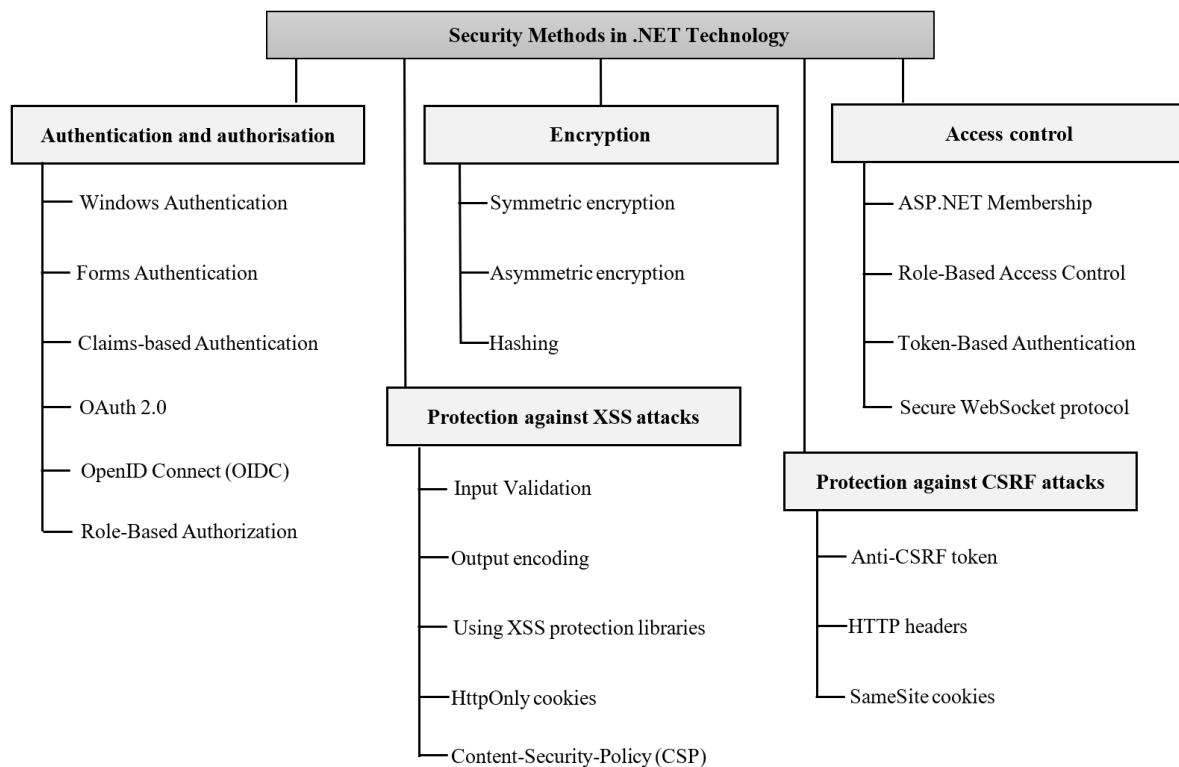


Figure 1: Security methods in .NET technology

OAuth 2.0 is a standard authentication protocol that allows web applications to access resources belonging to other users without having to reveal their passwords. This makes OAuth 2.0 a secure and convenient solution for authorising users in web applications. It is designed to provide controlled access to protected resources. OAuth 2.0 issues access tokens to third-party applications after obtaining the user's consent. These tokens provide limited access to the user's resources hosted on the resource server. The protocol defines different roles, such as client, resource owner, resource server and authorisation server, and different types of provisioning, such as authorisation code, implicit, client credentials and resource owner credentials, designed for different types of clients and scenarios. OpenID Connect (OIDC) is an authentication layer extension to the OAuth 2.0 protocol. While OAuth 2.0 focuses on authorisation, OIDC extends its functionality by adding an authentication component. One of the key aspects of OIDC is the concept of an ID token, which is not included in the OAuth 2.0 standard. This token contains information about the user's identity and is used in conjunction with an OAuth access token. The OIDC standardises the process of user identity verification and the mechanisms by which clients can request and receive data about authenticated sessions and end users [16].

Role-based authorisation is a method that uses roles to define user access rights to resources. Users are assigned roles, and each resource is assigned a set of permissions available to each role.

Role-based security can also be used when an application requires multiple approvals to perform an action. This might be the case in a procurement system where any employee can create a purchase request, but only a purchasing agent can turn that request into a purchase order that can be sent to the supplier [17].

Encryption is an important aspect of web application security, protecting sensitive data from unauthorised access [18–21]. .NET provides several symmetric and asymmetric encryption algorithms as well as hashing methods to protect data [22].

Symmetric encryption uses the same secret key to encrypt and decrypt data. It offers speed and efficiency, making it ideal for encrypting large amounts of data. In the symmetric cryptography mechanism, managed classes use a special stream called a `CryptoStream`. This stream encrypts the data entered into it. When the `CryptoStream` class is initialised, a managed stream is used that implements the `ICryptoTransform` interface, which corresponds to the cryptographic algorithm, and the `CryptoStreamMode`, which determines the type of access to the `CryptoStream` [10, 11].

Asymmetric encryption uses a pair of keys: a public key, which can be distributed publicly, and a private key, which is kept secret. The public key is used for encryption and the private key is used for decryption. To achieve this goal, .NET provides the `RSA` class. The `RSA` class is used to implement the asymmetric RSA algorithm. The security of the RSA cryptographic method is based on a computational problem, namely the decomposition of large integers into their prime factors [23].

Hashing converts data into a fixed size (hash) using a one-way function. It is impossible to recover the original data from the hash, making it useful for data integrity and authentication [24, 25]. When using ASP.NET Core Identity, the user provides his or her ID (user name) and password, along with other required data, when registering with the application. The application then generates a hash of the password and stores it in the database along with information about the user [26].

Access control is an important part of any web application's security system. It ensures that only authorised users have access to certain functions and data. .NET provides several access control mechanisms that allow developers to flexibly configure the security of their applications [27].

ASP.NET Membership provides a basic user management and authentication system for Web applications. It allows you to create user accounts, store user information, authenticate users, and manage their permissions. ASP.NET Membership includes features such as registering new users, authenticating registered users with a username and password, recovering passwords, and managing roles and access permissions. This allows developers to quickly and efficiently implement an authentication and authorisation system in their Web applications, providing a high level of security and access control [28].

RBAC (Role-Based Access Control) is a widely used access control model that uses roles and permissions to determine who has access to what resources. Employees only have access to the information they need to do their jobs effectively. Access can be based on a number of factors, such as authority, responsibility and professional competence. In addition, access to computer resources can be limited to specific tasks, such as the ability to view, create or modify a file [13].

Token-based authentication uses tokens to authenticate users. A token is a cryptographically signed packet of data that contains information about the user and their privileges. A user receives a token after successful authentication and uses it to access a web application. In the context of access control, servers use authentication tokens to verify the identity of a user, API, computer, or other server. Tokens can be physical (such as a USB key) or digital (a computer message or digital signature) [29].

WebSocket is a network communication protocol that allows .NET Web applications to establish two-way connections with a server. The WebSocket protocol in ASP.NET Core provides a two-way, persistent communication channel over TCP. This connection can be used to send and receive data in real time, making it ideal for chat rooms, online games, and other dynamic Web applications. Using WebSockets over HTTP/2 takes advantage of innovative features such as

header compression; multiplexing, which effectively reduces the time and resources required to send multiple requests to the server [30].

XSS attacks are a dangerous type of web attack that allows attackers to inject malicious JavaScript code into web pages that users view. This code can be used to steal confidential information, redirect users to phishing sites, or perform other malicious actions. In an XSS attack, an attacker uses a web application to send malicious code, usually in the form of a script, on the client side to another user who is unaware of it. The end user's browser may execute the malicious scripts, believing them to be safe, which can result in the malicious scripts gaining access to session tokens, cookies and other sensitive information stored by the browser and used on the website in question. These malicious scripts can change the content of HTML pages. The malicious content that is delivered to the browser usually consists of JavaScript segments, but can also include HTML, Flash, or any other type of code designed to perform malicious actions. Vulnerabilities that allow these attacks to occur are common and can occur anywhere a web application uses user input without encrypting or validating it. .NET provides several methods to protect against XSS attacks that help developers build secure web applications [31].

The first step in countering XSS attacks is to validate all user input before it is displayed on a web page. This can be done in a number of ways, including input validation and output encryption. Input validation involves checking user input for malicious code. If malicious code is detected, the data is rejected and the user is notified of the error. Input validation involves analysing the data for compliance with certain criteria. For example, if the user is asked to enter an email address, the check will include checking for the presence of the "@" symbol and the presence of a domain name. If the input data is incompatible with these criteria, it will be rejected. Output encoding is the transformation of user input so that it is not interpreted as HTML or JavaScript code. This can be achieved using a variety of techniques, such as HTML escaping and JavaScript encoding [32].

There are many .NET libraries (e.g. the XSS Prevention Library) that are specifically designed to protect against XSS attacks. These libraries provide out-of-the-box functions for output scraping, input validation, and other XSS security-related tasks. Anti-XSS Library is a library that provides a wide range of functions for output scanning, input validation, HTML and URL sanitisation, and detection and prevention of XSS attacks. OWASP Encoder is a library developed by the Open Web Application Security Project (OWASP) that provides functions for encoding output in various formats, including HTML, JavaScript, CSS, and XML. Microsoft Anti-XSS is a library from Microsoft that provides functions for escaping output in HTML, JavaScript, and other formats, and for preventing XSS attacks based on rendered data. These libraries help developers ensure the security of their applications by automating the process of protecting against potential attacks on web applications [31].

HttpOnly cookies are a type of cookie that is sent and received via the HTTP protocol. They are separate from other types of cookies and cannot be read or modified using the browser's document.cookie API. This feature increases your level of security by reducing your vulnerability to attacks aimed at manipulating cookies through malicious scripts. For example, if an attacker tries to use XSS to access document.cookie values and send them to a malicious server, the attempt will fail because HTTP-only cookies cannot be accessed through document.cookie. Furthermore, attempts to modify or delete HTTP-only cookies will also fail because the browser will ignore such changes [32].

Using a Content Security Policy (CSP) helps prevent malicious JavaScript code from being loaded from untrusted sources that can be used for XSS attacks. A CSP is an HTTP header that allows .NET web applications to specify which sources (domains, URLs) can load resources (scripts, styles, images, and so on) on a page. After defining a content security policy, web application developers can restrict the types of content that can be loaded and executed on a page. This helps prevent the execution of malicious scripts injected by attackers. The CSP provides several directives, such as "default-src", "script-src", "style-src", "img-src", "frame-src" and "connect-src", which allow you to configure these restrictions specifically for different types of content [33].

CSRF is a dangerous type of web attack that allows attackers to force users to perform unwanted actions on a website they are authorised to visit. CSRF attacks target functions that change the state of the server, such as changing personal information or making payments. Since the attacker does not need to receive a response to the request, but the victim does, the attacks aim to change the state. An attacker can use CSRF to obtain the victim's personal information by forcing them to enter it into a web application through a special request. Using social engineering, such as sending special links via email or chat, the attacker forces users to perform unwanted actions. If the victim is a regular user, a successful CSRF attack can trick them into performing state change requests, such as transferring funds or changing contact information. If the target is an administrator account, CSRF can lead to the compromise of the entire web application. .NET provides several methods of protection against CSRF attacks to help developers create secure web applications. An anti-CSRF token is a method where a unique CSRF token is generated for each user and attached to all forms and requests. The token is verified on the server to ensure that the request is being sent by an authorised user. Anti-CSRF tokens are randomly generated unique values embedded in HTML forms or web application URLs. These tokens are associated with a user's session and are used to authenticate subsequent requests sent on that user's behalf. The main purpose of anti-CSRF tokens is to ensure that all requests coming from the browser are legitimate and have not been tampered with by attackers. There are several HTTP headers that can be used to protect against CSRF attacks, such as X-CSRF-Token and X-Forwarded-For. These headers provide additional information about the request that can be used to verify it. The X-CSRF-Token header is used to send a CSRF token from the client to the server. The server must verify this token to ensure that the request is coming from a legitimate user. The X-Forwarded-For header is used to track the user's IP address when a request is forwarded through a proxy server. This can be useful in preventing CSRF attacks, which rely on spoofing the referrer. There is also the Origin header, which contains the URL of the origin of the request. The server can use this information to verify that the request is coming from the expected source. Using these headers can increase the security of web applications and prevent CSRF attacks. Reduces the risk of CSRF attacks using SameSite cookies [34]. SameSite is a cookie attribute that tells the browser when to send cookies along with requests to third-party websites. It helps prevent cookies containing authentication tokens or other sensitive data from being sent when you follow links from other websites. Possible values for this attribute are Lax, Strict, or None. For example, if a website like GitHub uses the Strict value, and a user tries to follow a link to a private GitHub project posted on a company forum or in an email, the user will not be able to access the project because GitHub will not receive a session cookie [35].

Table 1

Advantages and disadvantages of .NET security methods

#	Method of protection	Advantages	Disadvantages
1. Authentication and authorisation			
1	Windows Authentication	Easy to implement and configure; high level of security due to integration with Windows; no need to store passwords on the client side	Limited configuration and control of authentication, not suitable for independent web systems
2	Forms Authentication	Easy to implement; flexible as login and error pages can be customised; does not require Active Directory	Vulnerable to brute-force and phishing attacks unless additional security measures are taken; risk of password leakage; requires additional configuration

3	Claims-based Authentication	Allows you to include additional user attributes in the authentication token, which allows for more flexible access control; the ability to implement a single sign-on between applications	Requires additional development to integrate and manage user applications; may be difficult for users to understand tokens
4	OAuth 2.0	Safe and reliable; user-friendly; suitable for distributed systems	Complexity of implementation; dependence on third-party services; vulnerability to certain attacks
5	OpenID Connect (OIDC)	OIDC provides web applications with more user information, such as name, email address, and profile picture; OIDC simplifies OAuth 2.0 implementation; OIDC uses JWT to protect user information	OIDC requires OAuth 2.0 implementation, not all IdPs support OIDC
6	Role-Based Authorization	Easy to manage access to resources; reduces the burden on administrators; suitable for large systems	Requires additional configuration; may be difficult to implement; not suitable for simple websites

2. Encryption

7	Symmetric encryption	Fast and efficient data encryption and decryption process	Requires secure transmission and storage of a shared key for encryption and decryption
8	Asymmetric encryption	Higher security because a pair of keys is used, one of which remains private	Data exchange speed is reduced compared to symmetric encryption
9	Hashing	Irreversible process that guarantees safe data recovery	Vulnerable to brute-force attacks as there is no way to decrypt the hashed result

3. Access control

10	ASP.NET Membership	Flexibility (allows you to set up different access levels for different users); scalability; integration with other .NET components	Can be difficult to implement on large projects, and a database needs to be managed
11	Role-Based Access Control	Easy to add, remove, and change roles and permissions; suitable for large systems with many users and resources; to create complex access rules based on various factors	Requires additional configuration and setup; complex access rules can be difficult to understand and manage; may be overkill for small systems
12	Token-Based Authentication	Can be used with different authentication providers and protocols; suitable for large systems; tokens are digitally signed, which guarantees their reliability; authentication and authorisation are handled separately, making the system more flexible	Requires additional configuration and setup; users may need to understand how tokens work; not all websites and authentication providers support token-based authentication

13	Secure WebSocket protocol	WebSocket provides low latency; uses fewer resources than traditional HTTP requests, making it more efficient for streaming data efficient for streaming data	Complexity of implementation: WebSocket requires more development knowledge; WebSocket is not compatible with many browsers and servers WebSocket is not compatible with many browsers and servers
4. Protection against XSS attacks			
14	Input Validation	Effective against XSS attacks; easy to implement; can be used to protect different types of data	Can be difficult to identify all potentially dangerous symbols and designs
15	Output encoding	Effective against XSS attacks; easy to implement; can be used to protect different types of data	Can make the output unreadable; can lead to formatting issues
16	Using XSS protection libraries	Reduces the risk of XSS vulnerabilities; saves developers time and effort	Can make the code more complex; a suitable library must be selected and integrated
17	HttpOnly cookies	Using the HttpOnly attribute for cookies in the browser, which prevents JavaScript from accessing these cookies; easy to implement	Some older browsers do not support HttpOnly cookies; does not protect against other types of XSS attacks; may make it difficult to access cookies using JavaScript on the client side
18	Content-Security-Policy (CSP)	A powerful method of protecting against XSS attacks; can protect against other types of attacks; provides flexible control over what resources are allowed to be uploaded to a web page	Difficult to implement: needs to be carefully configured to avoid breaking the page; not all browsers support CSP

Analysis of the different authentication methods has shown that each has its own benefits and limitations. For example, Windows Authentication allows users to be identified using their Windows account, while Forms Authentication provides more flexibility but requires additional security attention. Therefore, it has been found that using a combination of different security methods is the key to ensuring a high level of security for .NET web applications. For example, to protect against XSS attacks, it is recommended that you use input validation, output encoding, HttpOnly cookies, and other measures described in this section. In addition, it is important to continually update and improve security practices based on the latest trends in the field.

To improve security in .NET, it is recommended that you use a comprehensive approach that combines various security methods, such as authentication, authorisation, encryption, access control, and intrusion prevention. It is also important to regularly update the software and components used in the application to fix identified vulnerabilities [36, 37]. Monitoring and auditing systems should be used to detect and analyse abnormal user activity and potential security threats. It is also necessary to set criteria for password complexity and use two-factor authentication for additional security [10].

Thus, the integration of these protective measures enables the formation of a unified security concept. A comprehensive model for implementing security methods that ensures the creation of a

secure web application using .NET technology can therefore be represented as a generalised security function S , which depends on a set of implemented security mechanisms.

$$S = f(M_1, M_2, \dots, M_n) \quad (1)$$

where M_i is the i^{th} security method (for example, M_1 is Windows Authentication, M_2 is Forms Authentication, ..., M_{18} is Content-Security-Policy).

Each security method M_i is characterised by an individual efficiency $E(M_i)$, which is determined in the range from 0 to 1, where the value 1 corresponds to the maximum possible level of protection.

The effectiveness indicator $E(M_i)$ should be considered as a functional dependence on a set of parameters, among which the key ones are the complexity of implementation and the coverage of vulnerabilities. The complexity of implementation reflects the time, labour and resource costs required to implement the relevant security mechanism, while vulnerability coverage characterises the proportion of potential threats neutralised by this method.

The generalised security level of system S is defined as the weighted sum of the effectiveness of all implemented methods, with the weighting coefficient w_i reflecting the relative importance of each method in ensuring a comprehensive protection strategy.

$$S = \sum_{i=1}^{n_i} w_i \times E(M_i). \quad (2)$$

The application of this approach provides the opportunity to quantitatively assess the level of security of an information system, which, in turn, contributes to increasing the objectivity of the analysis of its security. In addition, this model makes it possible to identify weaknesses through a comparative analysis of methods with low $E(M_i)$ values, as well as to plan investments in the implementation of individual security mechanisms in a reasonable manner, demonstrating their impact on the overall S indicator.

4. Conclusions

The security methods considered in .NET technology are key components in ensuring the reliability and security of software. The study analysed and described in detail various security approaches, such as authentication and authorisation using various mechanisms, including Windows Authentication, Forms Authentication, Claims-based Authentication, OAuth 2.0, OpenID Connect and Role-Based Authorisation. Encryption aspects were also explored, including symmetric and asymmetric encryption and hashing. ASP.NET membership access control, role-based access control, token-based authentication, and the secure WebSocket protocol were also covered.

Each of these methods has its own advantages and disadvantages, but by combining them we can create a comprehensive security system that ensures the highest level of security for .NET software. For example, different authentication and authorisation methods allow us to flexibly adapt the security system to the specific needs of the project, and protection measures against XSS attacks (Input Validation, Output Encoding, XSS protection libraries, HttpOnly cookies, Content Security Policy) and CSRF attacks (Anti-CSRF token, HTTP headers, SameSite cookies) help us to prevent various software vulnerabilities.

Thus, the conclusions of the study confirm that the implementation and use of various security methods in .NET technology is a key element for creating reliable and secure software. Careful analysis and selection of optimal security methods will help ensure the highest level of data and resource protection on the .NET platform, which is critical to the success of any project.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

- [1] O. Mykhaylova, M. Korol, R. Kyrychok, Research and Analysis of Issues and Challenges in Ensuring Cyber Security in Cloud Computing, in: *Cybersecurity Providing in Inf. and Telecomm. Systems II*, vol. 3826 (2024) 30–39.
- [2] Y. Martseniuk, et al., Automated Conformity Verification Concept for Cloud Security, in: *Cybersecurity Providing in Inf. and Telecomm. Systems, CPITS*, vol. 3654 (2024) 25–37.
- [3] Y. Martseniuk, et al., Universal Centralized Secret Data Management for Automated Public Cloud Provisioning, in: *Cybersecurity Providing in Inf. and Telecomm. Systems II*, vol. 3826 (2024) 72–81.
- [4] A. Ilyenko, et al., Practical Aspects of Using Fully Homomorphic Encryption Systems to Protect Cloud Computing, in: *Cybersecurity Providing in Inf. and Telecomm. Systems II*, vol. 3550 (2023) 226–233.
- [5] V. Shapoval, et al., Automation of Data Management Processes in Cloud Storage, in: *Cybersecurity Providing in Inf. and Telecomm. Systems, CPITS*, vol. 3654 (2024) 410–418.
- [6] Y. Martseniuk, et al., Research of the Centralized Configuration Repository Efficiency for Secure Cloud Service Infrastructure Management, in: *Cybersecurity Providing in Inf. and Telecomm. Systems*, vol. 3991 (2025) 260–274.
- [7] O. Vakhula, I. Opirskyy, O. Mykhaylova, Research on Security Challenges in Cloud Environments and Solutions based on the “Security-as-Code” Approach, in: *Cybersecurity Providing in Inf. and Telecomm. Systems*, vol. 3550 (2023) 55–69.
- [8] altexsoft.com. <https://www.altexsoft.com/blog/the-good-and-the-bad-of-net-framework-programming/>
- [9] C#. Kontseptsii ta syntaksys. Navch. Posibnyk. Lviv, Vydavnychiy tsentr LNU imeni Ivana Franka, 2006. https://ami.lnu.edu.ua/wp-content/uploads/2017/05/C_sharp.pdf
- [10] support.microsoft.com. <https://support.microsoft.com/en-us/topic/asp-net-security-overview-7c8562d3-7bea-306c-4c78-98dd6c6993b3>
- [11] medium.com. <https://medium.com/@kerimkkara/authentication-and-authorization-in-asp-net-core-a-comprehensive-guide-dfb8fb806ac7>
- [12] learn.microsoft.com. <https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/windows-authentication-overview>
- [13] P. Petriv, I. Opirskyy, N. Mazur, Modern Technologies of Decentralized Databases, Authentication, and Authorization Methods, in: *Cybersecurity Providing in Inf. and Telecomm. Systems II*, vol. 3826 (2024) 60–71.
- [14] c-sharpcorner.com. <https://www.c-sharpcorner.com/UploadFile/fa9d0d/forms-authentication-in-Asp-Net>
- [15] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-8.0>
- [16] learn.microsoft.com. <https://learn.microsoft.com/uk-ua/power-pages/security/authentication/oauth2-settings>
- [17] learn.microsoft.com. <https://learn.microsoft.com/uk-ua/aspnet/core/security/authorization/roles?view=aspnetcore-3.1>
- [18] V. Sokolov, P. Skladannyi, H. Hulak, Stability Verification of Self-Organized Wireless Networks with Block Encryption, in: *5th Int. Workshop on Computer Modeling and Intelligent Systems*, vol. 3137 (2022) 227–237.

- [19] M. Iavich, et al., Classical and Post-Quantum Encryption for GDPR, in: *Classic, Quantum, and Post-Quantum Cryptography*, vol. 3829 (2024) 70–78.
- [20] R. Chernenko, et al., Encryption Method for Systems with Limited Computing Resources, in: *Cybersecurity Providing in Information and Telecomm. Systems*, vol. 3288 (2022) 142–148.
- [21] A. Ilyenko, et al., Practical Aspects of Using Fully Homomorphic Encryption Systems to Protect Cloud Computing, in: *Cybersecurity Providing in Information and Telecomm. Systems II*, vol. 3550 (2023) 226–233.
- [22] learn.microsoft.com. <https://learn.microsoft.com/en-us/dotnet/standard/security/cryptographic-services>
- [23] learn.microsoft.com. <https://learn.microsoft.com/en-us/dotnet/standard/security/encrypting-data>
- [24] D. Shevchuk, et al., Designing Secured Services for Authentication, Authorization, and Accounting of Users, in: *Cybersecurity Providing in Information and Telecommunication Systems II*, vol. 3550 (2023) 217–225.
- [25] Y. Shcheblanin, et al., Research of Authentication Methods in Mobile Applications, in: *Cybersecurity Providing in Information and Telecomm. Systems*, vol. 3421 (2023) 266–271.
- [26] learn.microsoft.com. <https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.hashalgorithm?view=net-8.0>
- [27] learn.microsoft.com. <https://learn.microsoft.com/en-us/dotnet/api/system.security.accesscontrol?view=net-8.0>
- [28] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/web-forms/overview/moving-to-aspnet-20/membership>
- [29] medium.com. <https://medium.com/@microclip.lakeesha/net-core-6-token-based-authentication-and-middleware-43a5fc86089a>
- [30] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/websockets?view=aspnetcore-8.0>
- [31] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/core/security/cross-sitescripting?view=aspnetcore-8.0>
- [32] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/web-pages/overview/ui-layouts-and-themes/validating-user-input-in-aspnet-web-pages-sites>
- [33] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/core/blazor/security/content-security-policyview=aspnetcore-8.0>
- [34] Y. Kostiuk, et al., Models and Algorithms for Analyzing Information Risks during the Security Audit of Personal Data Information System, in: *3rd Int. Conf. on Cyber Hygiene and Conflict Management in Global Information Networks*, vol. 3925 (2025) 155–171.
- [35] learn.microsoft.com. <https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-8.0>
- [36] Y. Kostiuk, et al., Architecture of the Software System of Confidential Access to Information Resources of Computer Networks, in: *Cyber Security and Data Protection*, vol. 4042 (2025) 37–53.
- [37] I. Tyshyk, H. Hulak, Testing an Organization's Information System for Unauthorized Access, in: *Cybersecurity Providing in Inf. and Telecomm. Systems*, vol. 3826 (2024) 17–29.