Семенюк А. В.,
здобувач третього (освітньо-наукового) рівня вищої освіти,
Вінницький національний технічний університет, м. Вінниця
Богач І. В.,
канд. техн. наук, доцент,
професор кафедри автоматизації та інтелектуальних інформаційних технологій,
Вінницький національний технічний університет, м. Вінниця

УДК 004.056.5
DOI: https://doi.org/10.64076/ihrc251010.05

CC BY

КІБЕРБЕЗПЕКА ВЕБ-ЗАСТОСУНКІВ: ВДОСКОНАЛЕННЯ МЕТОДІВ ЗАХИСТУ

Вступ. Проблема забезпечення кібербезпеки веб-застосунків набуває все більшої актуальності у сучасних умовах цифрової економіки. Веб-застосунки обробляють чутливі дані користувачів (особисту інформацію, фінансові дані тощо), і успішні атаки на них можуть призвести до витоків даних, фінансових втрат та підриву репутації організацій. Згідно з останніми дослідженнями, кібератаки на веб-застосунки зростають тривожними темпами – середньодобова кількість атак перевищує 62 млн (на 12,6% більше, ніж торік). Середня вартість одного витоку даних у світі сягнула 4,35 млн доларів у 2022 році, що є рекордним показником. Ці факти підкреслюють необхідність удосконалення методів захисту веб-застосунків для протидії сучасним кіберзагрозам.

Веб-загрози є численними та різноманітними. Багато з них систематизовано в переліку OWASP Top 10[1] – загальновизнаному стандарті, що окреслює найбільш критичні ризики безпеки веб-додатків. Актуальна версія OWASP Top 10 (2021) визначає, зокрема, такі провідні загрози як порушення контролю доступу (Broken Access Control), криптографічні несправності (Cryptographic Failures) та ін'єкції (Injection). В категорію ін'єкцій входять класичні атаки, як-от SQL-ін'єкція та міжсайтовий скриптинг (XSS). XSS упродовж років лишається одним із найпоширеніших видів вразливостей – за оцінками OWASP, ця проблема виявляється у двох третинах веб-застосунків і може призвести до виконання шкідливого коду в браузері жертви (крадіжки сесій, даних тощо). Інша небезпечна атака – міжсайтове підроблення запитів (CSRF), коли зловмисник змушує користувача несвідомо виконати несанкціоновану дію в системі, в якій

той автентифікований. До критичних загроз також належать помилки автентифікації та управління сесіями, що дають змогу захопити облікові записи, вразливості конфігурації, використання вразливих компонентів тощо.

Таким чином, забезпечення безпеки веб-застосунків потребує комплексного підходу із врахуванням актуальних атак і методів захисту. У даній роботі здійснено огляд існуючих методів захисту веб-застосунків та запропоновано напрями їх вдосконалення з урахуванням новітніх технологій розробки (Node.js, React) та рекомендацій спільноти OWASP станом на 2023 рік.

Метою дослідження є підвищення ефективності захисту веб-застосунків шляхом удосконалення існуючих методів кібербезпеки. Для досягнення цієї мети проаналізовано розповсюджені вразливості веб-застосунків і засоби протидії їм, визначено недоліки традиційних методів захисту та розроблено рекомендації щодо їх поліпшення з урахуванням специфіки сучасних веб-технологій.

Об’єктом дослідження є процеси та методи забезпечення кібербезпеки веб-застосунків, включаючи механізми виявлення й запобігання атакам на веб-застосунки, а також інструменти захисту, інтегровані в сучасні фреймворки (React, Node.js) і платформи розробки веб-додатків.

Наукова новизна одержаних результатів полягає у формуванні вдосконаленого підходу до захисту веб-застосунків на основі поєднання найкращих практик та новітніх технологій. Зокрема, вперше узагальнено досвід застосування політик Content Security Policy, сучасних механізмів автентифікації/авторизації та "безпечного за замовчуванням" програмування (як-от автоматичне екранування вихідних даних у фреймворку React) для нейтралізації загроз типу XSS, CSRF, SQL-ін'єкцій тощо[6]. Запропоновано рекомендації з впровадження політики "білого списку" при обробці даних і налаштуванні веб-застосунків, інтеграції засобів захисту на всіх етапах життєвого циклу розробки (DevSecOps) та використання спеціалізованих бібліотек безпеки для Node.js[4,5]. Такий підхід підвищує рівень захищеності веб-застосунків і стійкість до актуальних кіберзагроз.

Практичне значення одержаних результатів полягає в тому, що розроблені рекомендації можуть бути безпосередньо використані у процесі створення та підтримки веб-застосунків. Використання запропонованих методів (впровадження CSP, налаштування secure headers, застосування багатофакторної автентифікації, регулярний аналіз вразливостей тощо) дозволить зменшити ризик успішних атак на веб-застосунки та запобігти витокам даних. Результати роботи можуть слугувати підґрунтям для оновлення корпоративних стандартів безпеки розробки, проведення навчань для веб-розробників щодо безпечного кодування, а також для подальших наукових досліджень у галузі кібербезпеки веб-систем.

Існуючі методи захисту веб-застосунків. За останні десятиліття розроблено низку стандартних підходів і рекомендацій, спрямованих на захист веб-застосунків від відомих типів атак. Основою безпеки є принцип "безпечного за дизайном": системи повинні проєктуватися з урахуванням потенційних загроз, щоб запобігти вразливостям на етапі розробки. Низка фундаментальних методів захисту вже стала галузевим стандартом:

Валідація та фільтрація вводу. Усі дані, що надходять від користувача (поля форм, параметри URL тощо), мають перевірятися та очищуватися перед обробкою на сервері. Вхідні дані слід допустимим чином обмежувати (використовуючи білі списки дозволених значень) та екранувати спеціальні символи. Такий підхід попереджає атаки ін'єкцій, зокрема SQL-ін'єкції, XSS та ін’єкції команд. Наприклад, щоб уникнути SQL-ін'єкції, розробники застосовують параметризовані запити або ORM-засоби, які автоматично екранують параметри запитів до бази даних.

Кодування вихідних даних (output escaping). Перед відображенням будь-яких даних користувача у браузері застосовуються механізми екранування HTML/JavaScript[2]. Це гарантує розділення виконуваного коду і даних та запобігає виконанню вбудованих скриптів, нейтралізуючи XSS. Сучасні фреймворки, такі як React.js, реалізують автоматичне екранування виводу за замовчуванням (якщо тільки не використовується небезпечний механізм на кшталт dangerouslySetInnerHTML)[2,3], що значно ускладнює здійснення XSS-атак. Використання фреймворків з вбудованими засобами безпеки є однією з найкращих практик розробки безпечних веб-застосунків.

Механізми автентифікації та контролю доступу. Надійна автентифікація користувачів та авторизація їх дій є критичною для захисту від захоплення облікових записів та несанкціонованого доступу. Рекомендовано впроваджувати багатофакторну автентифікацію, складні вимоги до паролів, обмеження кількості спроб входу (для захисту від brute-force) тощо. Управління сесіями повинно бути належним чином захищене: сесійні ідентифікатори мають бути достатньо довгими та випадковими, передаватися тільки по захищеному каналу, супроводжуватися прапорцями HttpOnly та Secure в cookies. Система доступу повинна будуватися за принципом найменших привілеїв – кожному користувачу та компоненту надаються лише ті права, які необхідні для виконання його функцій. Правильна реалізація контролю доступу допомагає запобігти таким загрозам як Broken Access Control, що займає перше місце в OWASP Top 10[1].

Шифрування даних та захищені комунікації. Одним з базових методів є використання шифрування для конфіденційних даних. Всі чутливі дані (паролі, фінансова інформація) повинні зберігатися у зашифрованому або гешованому

вигляді. Сучасні алгоритми гешування паролів (bcrypt, scrypt, Argon2) [4,5] значно ускладнюють отримання паролю навіть у випадку компрометації бази даних. Для передачі даних між клієнтом і сервером обов'язково слід використовувати протокол HTTPS (TLS), що шифрує трафік і запобігає його перехопленню та підробленню. Використання TLS сертифікатів від надійних центрів сертифікації та активація HSTS (HTTP Strict Transport Security) заголовку забезпечують, що користувацькі з'єднання завжди залишаються захищеними.

Безпечна обробка помилок і логування. Веб-застосунок повинен коректно обробляти помилки, не розкриваючи внутрішню інформацію (стектрейси, повідомлення СУБД тощо) користувачам, оскільки така інформація може допомогти зловмисникам. Натомість детальна інформація про помилки має фіксуватися у внутрішніх журналах. Ведення журналів безпеки дозволяє виявляти спроби атак та інциденти. Системи моніторингу та аналізу логів (SIEM) можуть оперативно сповіщати адміністраторів про підозрілі дії (наприклад, численні невдалі спроби входу, SQL-ін'єкції, виявлені у рядках запитів тощо). Таким чином, належна обробка виключень і аудит дій користувачів є необхідними складовими кіберзахисту[5].

Регулярне оновлення та патчинг. Вразливості часто виникають у використовуваних веб-застосунком компонентах – фреймворках, бібліотеках, серверному ПЗ. Тому важливо своєчасно оновлювати програмне забезпечення та встановлювати патчі безпеки, щойно вони стають доступними. Це стосується як бекенд-частини (напр. оновлення Node.js та залежних npm-пакетів) [4,5], так і фронтенду (оновлення бібліотек React, Angular тощо). Багато відомих інцидентів сталися через експлуатацію вразливостей у старих версіях компонентів, що вже мали виправлення. Практика постійного оновлення і менеджменту вразливостей залежностей (включаючи перевірку бібліотек на наявність відомих CVE) є обов'язковою складовою безпеки (ця проблема відображена як "Використання компонентів із відомими вразливостями" в OWASP Top 10).

Мережевий захист та моніторинг. Існуючі методи включають розгортання Web Application Firewall (WAF) перед веб-застосунком для фільтрації шкідливого трафіку за відомими шаблонами атак. WAF може блокувати типові SQL-ін'єкції, XSS, а також захищати від деяких DDoS-атак на рівні застосунку. Додатково використовуються системи виявлення та запобігання вторгнень (IDS/IPS), що відстежують підозрілу активність у мережевому трафіку та на хості. Хоча традиційні мережеві засоби (фаєрволи, IPS) не можуть повністю захистити від логічних веб-атак на рівні застосунку, вони є важливим елементом багаторівневого захисту (Defense in Depth).

Аудит безпеки та тестування на проникнення. Регулярні перевірки безпеки є обов'язковими для виявлення слабких місць. Автоматизовані сканери вразливостей допомагають проаналізувати веб-додаток на наявність відомих проблем (SQL-ін'єкції, XSS, неправильні налаштування тощо). Разом з тим, penetration testing (тестування на проникнення) вручну або за допомогою спеціалістів з кібербезпеки дозволяє імітувати дії зловмисника та виявити логічні уразливості, які не завжди знаходяться автоматикою. За результатами тестувань розробники отримують звіти з описом знайдених проблем і рекомендаціями щодо їх усунення.

Перелічені існуючі методи складають базис захисту веб-застосунків. Однак постійна еволюція кіберзагроз вимагає не лише застосування цих підходів, але й їх подальшого вдосконалення та адаптації до сучасних умов, про що йдеться далі.

Вдосконалення існуючих методів. У сучасних веб-застосунках з'являються нові вектори атак та виклики, тому традиційні методи захисту потребують удосконалення. Нижче наведено напрями покращення безпеки веб-додатків з урахуванням новітніх технологій і підходів:

Впровадження політики "білого списку" та Content Security Policy. Класичні методи фільтрації часто базуються на принципі "чорного списку" (блокування відомих шкідливих шаблонів). Проте більш ефективним підходом є політика whitelist – заборона всього, що не дозволено явно. Прикладом є Content Security Policy (CSP) – політика безпеки контенту, яка задається через HTTP-заголовки і визначає, з яких джерел дозволено завантажувати ресурси (скрипти, стилі, зображення тощо). Правильно налаштована CSP значно ускладнює проведення XSS-атак, оскільки браузер блокуватиме завантаження виконуваного коду з неперевірених джерел[7]. CSP виступає додатковим рубежем захисту ("line of defense in depth") на випадок, якщо інші заходи (валідація даних, екранування) дали збій. До впровадження CSP слід підходити комплексно, тестуючи політики в режимі моніторингу (Content-Security-Policy-Report-Only) та поступово посилюючи правила. Крім CSP, концепція "білого списку" може застосовуватися й в інших аспектах: зокрема, жорстке обмеження доступних API у браузері (через заголовки Feature Policy / Permissions Policy) або явне перелічення дозволених перенаправлень та доменів для інтеграцій, щоб запобігти атакам шляхом підміни адрес.

Інтеграція безпеки в життєвий цикл розробки (DevSecOps). Традиційно безпекою займалися на фінальних етапах (або після впровадження системи), що часто призводило до запізнілого виявлення проблем. Сучасний підхід DevSecOps передбачає, що безпека інтегрується на всіх фазах розробки програмного забезпечення. Це означає: проведення threat modeling і аналізу ризиків ще на стадії проєктування; використання статичних аналізаторів коду (SAST) та

аналізаторів залежностей під час розробки для виявлення уразливостей; виконання автоматизованих тестів безпеки (DAST) на етапі CI/CD; впровадження безперервного моніторингу застосунку в продакшні. Такий підхід дозволяє виявляти та усувати вразливості на ранніх стадіях, коли їх виправлення менш затратне, а також створює культуру "безпечної розробки" в команді. В результаті зменшується кількість помилок безпеки у фінальному продукті і підвищується довіра користувачів до системи.

Використання спеціалізованих бібліотек та фреймворків безпеки. В середовищі Node.js існує широкий набір готових рішень для посилення безпеки веб-застосунків. Наприклад, популярний пакет Helmet для Express.js автоматично встановлює ряд HTTP-заголовків безпеки (XSS-Protection, X-Frame-Options, HSTS, Content-Type Options тощо), що допомагають запобігти поширеним атакам. Використання такого інструментарію значно спрощує реалізацію найкращих практик: достатньо один раз підключити middleware, щоб забезпечити стандартний набір захисних заголовків для всіх відповідей сервера. Крім того, OWASP спільнотою підтримуються проєкти на зразок OWASP Cheat Sheet Series – детальні рекомендації для розробників. Зокрема, Node.js Security Cheat Sheet містить перелік кроків для захисту Node-додатків: від уникнення блокування Event Loop, налаштування лімітів на розмір запитів, до правил безпечної роботи з пакунками та контролю викликів небезпечних функцій. Також започатковано проєкт OWASP Bulletproof React, метою якого є збір найкращих практик захисту для стеку React+Node.js і демонстрація реальних вразливостей та способів їх усунення на прикладах. Використання таких спеціалізованих інструментів і гайдів дозволяє розробникам врахувати максимальну кількість потенційних проблем безпеки та суттєво підвищує базовий рівень захищеності застосунку.

Удосконалення методів автентифікації та керування сесіями. З розвитком технологій зловмисники почали ширше використовувати методи обходу традиційної парольної автентифікації (фішинг, перехоплення одноразових паролів тощо). Тому вдосконалення потребують і засоби підтвердження особи користувача. Перспективними напрямами є впровадження біометричної автентифікації, апаратних токенів безпеки (ключі U2F/FIDO2) та систем single sign-on з надійними постачальниками ідентифікації. Для захисту веб-сесій доцільно розглядати механізми повторної верифікації (re-authentication) для критичних операцій, прив’язку сесії до клієнтського середовища (використання параметрів браузера, IP-адреси) та більше використання JSON Web Token (JWT) з коротким часом життя і обов’язковою перевіркою підпису на сервері. Всі ці міри знижують

ймовірність компрометації сесій і облікових записів навіть якщо окремі креденшіали були викрадені.

Протидія новим видам атак. Сучасні веб-застосунки повинні враховувати не лише класичні загрози, а й новітні техніки атак. Наприклад, поява категорії Server-Side Request Forgery (SSRF) як окремої небезпеки (включеної в OWASP Top 10 2021) вимагає додаткових перевірок при обробці сервером будь-яких URL чи запитів до зовнішніх ресурсів. Для запобігання SSRF-зловживанням рекомендується суворо обмежувати діапазон адрес, до яких сервер може звертатися, і виконувати валідацію форматів URL. Інший напрям – безпека API та мікросервісів, адже сучасні веб-застосунки часто побудовані як набір взаємодіючих API. Необхідно впроваджувати захист на рівні API (наприклад, використання OAuth2/OIDC для авторизації запитів між сервісами, стандарти OpenAPI/Swagger для специфікації та подальшого сканування безпеки API). Окрема увага – безпеці на стороні клієнта: поширення односторінкових застосунків (SPA) на React/Angular означає, що значна частина логіки працює в браузері, тому важливо захищати клієнтський код від підроблення (через ті ж CSP, інтеграцію підписів скриптів Subresource Integrity тощо) і враховувати можливість атак на ланцюжок поставок (supply chain attacks) через компрометацію сторонніх скриптів або npm-пакетів. Для зменшення ризиків supply chain рекомендується мінімізувати використання зовнішніх скриптів, регулярно перевіряти їх оновлення та використовувати дзеркала npm-репозиторіїв з перевіркою цілісності пакетів.

В цілому, вдосконалення методів захисту веб-застосунків зводиться до переходу від реактивної моделі ("латання" відомих дір) до проактивної, превентивної моделі захисту. Це досягається через глибоке впровадження принципів secure by design, використання сучасних інструментів для автоматизації безпеки та постійне підвищення обізнаності розробників щодо кіберзагроз.

Висновки. У даній статті проведено огляд поточних загроз кібербезпеці веб-застосунків та заходів протидії ним, а також запропоновано напрямки вдосконалення методів захисту. Проаналізовано типові атаки (XSS, CSRF, SQL-ін'єкції, SSRF тощо) та їх вплив на безпеку веб-додатків. Встановлено, що класичні методи захисту (валідація даних, контроль доступу, шифрування, тощо) залишаються необхідними, але мають доповнюватися новими підходами для ефективного протистояння сучасним загрозам. На основі рекомендацій OWASP і аналізу актуальних інцидентів обґрунтовано доцільність впровадження політик

безпеки контенту (CSP) та інших secure headers, використання принципу білого списку, інтеграції процесів безпеки в SDLC (DevSecOps) та застосування спеціалізованих засобів (як-от Helmet для Node.js, безпечні фреймворки на кшталт React). Результати роботи можуть бути використані для розробки більш захищених веб-застосунків, що особливо актуально в контексті поширення JavaScript-фреймворків (React, Node.js) та зростання числа кібератак на веб-ресурси. Подальші дослідження можуть бути спрямовані на експериментальну оцінку ефективності запропонованих підходів та розробку автоматизованих інструментів контролю безпеки в процесі розробки і експлуатації веб-систем.

Список використаних джерел

1. OWASP Top Ten. URL: https://owasp.org/www-project-top-ten (дата звернення: 23.10.2025).

2. React.js Security Guide: Threats, Vulnerabilities, and Ways to Fix Them in 2025. URL: https://relevant.software/blog/react-js-security-guide (дата звернення: 23.10.2025).

3. React JS – Security Best Practices. URL: https://relevant.software/blog/react-js-security-guide (дата звернення: 23.10.2025).

4. Learn Node.js Secure Coding Techniques and Best Practices. URL: https://github.com/lirantal/awesome-nodejs-security (дата звернення: 23.10.2025).

5. 7 Best Practices for Node.js Security in Production. URL: https://article.arunangshudas.com/7-best-practices-for-node-js-security-in-production-cd03a2dd9795 (дата звернення: 23.10.2025).

6. Testing for Cross Site Request Forgery. URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/05-Testing_for_Cross_Site_Request_Forgery (дата звернення: 23.10.2025).

7. Як забезпечити безпеку веб-додатків: найкращі практики та стратегії. URL: https://stfalcon.com/uk/blog/post/building-secure-web-applications-best-practices-and-strategies (дата звернення: 23.10.2025).