Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

Технологія блокчейн та розподілені системи

Лабораторна робота №3

“Дослідження безпечної реалізації та експлуатації децентралізованих додатків **”**

Виконали:
Студенти групи ФБ-42мп
Алькова Аліна, Легойда Юлія, Рябко Дмитро

Тема: Дослідження безпечної реалізації та експлуатації децентралізованих додатків.
Мета роботи: отримання навичок роботи із децентралізованими додатками та оцінка безпеки інформації при їх функціонуванні

Завдання для першого типу лабораторних робіт: дослідження вимог OWASP (безпека web-додатків) та складання аналогічних вимог для обраної системи децентралізованих додатків.

1. Дослідження вимог OWASP

OWASP(Top 10) — це стандарт, розроблений Open Web Application Security Project (OWASP), який визначає 10 найкритичніших вразливостей безпеки веб-додатків. Цей документ містить структурований огляд кожної категорії ризиків, їх опис, приклади вразливостей та потенційні наслідки. Він призначений для використання в аналізі безпеки децентралізованих додатків (dApps) у рамках командного завдання.

Категорії OWASP Top 10

№	Категорія	Опис	Приклад вразливості	Потенційні наслідки
A01	**Broken Access Control**	Недоліки в управлінні доступом дозволяють неавторизованим користувачам отримувати доступ до даних або функцій.	Користувач змінює ID у URL (наприклад, /user/123 на /user/456) і отримує доступ до даних іншого користувача.	Витік конфіденційної інформації, несанкціоновані дії, компрометація системи.
A02	**Cryptographic Failures**	Неправильне використання криптографії або її відсутність, що призводить до витоку даних.	Зберігання паролів у відкритому вигляді або використання застарілих алгоритмів шифрування (наприклад, MD5).	Витік даних користувачів, компрометація облікових записів.

A03	**Injection**	Введення неперевірених даних (наприклад, SQL, OS, LDAP) у систему, що дозволяє виконувати шкідливий код.	SQL-ін’єкція через поле введення: SELECT * FROM users WHERE id = '1' OR '1'='1'.	Витік даних, виконання несанкціонованих команд, повна компрометація бази даних.
A04	**Insecure Design**	Недоліки в архітектурі або дизайні додатку, які неможливо виправити лише технічними засобами.	Відсутність двофакторної аутентифікації в системі з чутливими даними.	Системні вразливості, які важко виправити без переробки дизайну.
A05	**Security Misconfiguration**	Неправильна конфігурація серверів, фреймворків або компонентів.	Увімкнені дебаг-режими на продакшен-сервері, що розкривають системну інформацію.	Несанкціонований доступ, витік даних, компрометація системи.
A06	**Vulnerable and Outdated Components**	Використання застарілих бібліотек або компонентів із відомими вразливостями.	Використання старої версії бібліотеки Apache Struts із відомою CVE-вразливістю.	Виконання шкідливого коду, повна компрометація системи.
A07	**Identification and Authentication Failures**	Проблеми з аутентифікацією або управлінням сесіями.	Слабкі паролі або незахищені сесійні токени, які можна вкрасти.	Несанкціонований доступ до облікових записів користувачів.

A08	**Software and Data Integrity Failures**	Недостатня перевірка цілісності програмного забезпечення або даних.	Використання непідписаних оновлень ПЗ або незахищених CI/CD-пайплайнів.	Встановлення шкідливого ПЗ, компрометація даних.
A09	**Security Logging and Monitoring Failures**	Відсутність або недостатнє логування та моніторинг подій безпеки.	Відсутність логів для відстеження спроб несанкціонованого доступу.	Неможливість виявити атаку або провести розслідування після інциденту.
A10	**Server-Side Request Forgery (SSRF)**	Додаток дозволяє відправляти запити до ненадійних ресурсів від імені сервера.	Зловмисник змушує сервер зробити запит до внутрішньої системи (наприклад, http://localhost/admin).	Доступ до внутрішніх ресурсів, витік даних.

Детальний опис категорій

A01: Broken Access Control - вразливості виникають, коли додаток не перевіряє належним чином права доступу користувачів. Це дозволяє зловмисникам отримувати доступ до даних або функцій, які мають бути обмеженими.

- **Приклад**: Користувач змінює параметр у запиті (наприклад, user_id=123 на user_id=124) і отримує доступ до профілю іншого користувача.
- **Рекомендації**: Використовувати принцип найменших привілеїв, перевіряти права доступу на сервері.

A02: Cryptographic Failures - проблеми з шифруванням, наприклад, використання слабких алгоритмів, ненадійне зберігання ключів або передача даних у незашифрованому вигляді.

- **Приклад**: Передача паролів через HTTP замість HTTPS.
- **Рекомендації**: Використовувати сучасні алгоритми шифрування (наприклад, AES-256, SHA-256), забезпечити безпечне зберігання ключів.

A03: Injection - зловмисник вводить шкідливі дані, які інтерпретуються системою як команди (наприклад, SQL, NoSQL, OS-команди).

- **Приклад**: Введення ' OR '1'='1 у поле логіну для обходу аутентифікації.
- **Рекомендації**: Використовувати параметризовані запити, екранування введених даних.

A04: Insecure Design - проблеми в дизайні системи, які не можна виправити лише технічними засобами (наприклад, відсутність захисту від повторного використання паролів).

- **Приклад**: Відсутність механізмів захисту від брутфорс-атак.
- **Рекомендації**: Впроваджувати принципи безпечного дизайну (Secure by Design) на етапі проєктування.

A05: Security Misconfiguration - некоректна конфігурація серверів, баз даних або бібліотек, що відкриває можливості для атак.

- **Приклад**: Увімкнення списків директорій на веб-сервері.
- **Рекомендації**: Використовувати автоматизовані інструменти для перевірки конфігурацій, відключати непотрібні функції.

A06: Vulnerable and Outdated Components - використання бібліотек або фреймворків із відомими вразливостями.

- **Приклад**: Застаріла версія jQuery із відомою XSS-вразливістю.
- **Рекомендації**: Регулярно оновлювати компоненти, використовувати інструменти для сканування залежностей (наприклад, Dependabot).

A07: Identification and Authentication Failures - проблеми з аутентифікацією, такі як слабкі паролі, незахищені сесії або відсутність двофакторної аутентифікації.

- **Приклад**: Сесійний токен передається через незахищений канал.
- **Рекомендації**: Впроваджувати двофакторну аутентифікацію, використовувати безпечні методи зберігання паролів (bcrypt).

A08: Software and Data Integrity Failures - відсутність перевірки цілісності даних або програмного забезпечення, що дозволяє зловмисникам модифікувати їх.

- **Приклад**: Завантаження непідписаного оновлення ПЗ.
- **Рекомендації**: Використовувати цифрові підписи, перевіряти хеші файлів.

A09: Security Logging and Monitoring Failures - відсутність належного логування подій безпеки або моніторингу атак.

- **Приклад**: Неможливість виявити повторні спроби входу через відсутність логів.

- **Рекомендації**: Впроваджувати централізоване логування, налаштувати сповіщення про підозрілі дії.

A10: Server-Side Request Forgery (SSRF) - додаток дозволяє серверу виконувати запити до ненадійних або внутрішніх ресурсів.

- **Приклад**: Зловмисник відправляє запит до http://internal-api/ через вразливе API.
- **Рекомендації**: Обмежувати домени, до яких сервер може робити запити, використовувати білі списки.

2. Аналіз особливостей децентралізованих додатків (dApps) ETHEREUM

Дослідження специфіки обраної системи децентралізованих додатків

Серед сучасних блокчейн-платформ, що підтримують децентралізовані додатки (dApps), особливу увагу привертають Ethereum, Solana та Polkadot. Кожна з них має унікальну архітектуру, механізми консенсусу, підхід до масштабування та екосистему. Розглядаючи специфіку цих систем, варто зосередитися на їхньому призначенні, технічних особливостях, перевагах та викликах, з якими стикаються розробники й користувачі.

Ethereum — піонер у сфері децентралізованих додатків. Його основною особливістю є підтримка смарт-контрактів на базі мови програмування Solidity. Ці контракти зберігаються та виконуються у віртуальній машині Ethereum (EVM), що гарантує передбачувану поведінку додатків у мережі. Ethereum підтримує повнофункціональне програмування з умовами, циклами, логікою тощо, що дозволяє реалізовувати складні бізнес-процеси. Основна проблема платформи — низька пропускна здатність та високі транзакційні збори, які загострюються в періоди високого навантаження. Для вирішення цієї проблеми активно розробляються рішення другого рівня (Layer 2), а перехід до Proof of Stake у межах Ethereum 2.0 має на меті підвищити масштабованість та енергоефективність мережі. Платформа залишається найбільш використовуваною серед dApps — особливо в секторах DeFi, NFT та DAO.

Solana, у свою чергу, позиціонує себе як високо масштабована блокчейн-платформа з надзвичайно високою швидкістю обробки транзакцій. На відміну від Ethereum, який обробляє лише кілька десятків транзакцій за секунду, Solana досягає показників у десятки тисяч транзакцій за секунду завдяки унікальному алгоритму консенсусу Proof of History (PoH), який забезпечує хронологічний порядок подій. Ця технологія дозволяє мінімізувати час узгодження між вузлами мережі, прискорюючи виконання транзакцій. Проте швидкість досягається ціною централізації — мережа покладається на високопродуктивні вузли, що ускладнює повноцінну участь для звичайних користувачів. Крім того, Solana неодноразово стикалася з технічними проблемами,

включаючи тимчасові зупинки мережі, що ставить під сумнів її надійність у критичних додатках.

Polkadot пропонує інноваційний підхід до побудови блокчейн-екосистеми. Його архітектура передбачає наявність основного ланцюга (Relay Chain), який забезпечує безпеку та консенсус, та підключених парачейнів — спеціалізованих блокчейнів, які можуть мати власну логіку та структуру даних. Така модель дозволяє реалізовувати міжланцюгову взаємодію, гнучкість у дизайні та масштабованість. Парачейни можуть бути оптимізовані для конкретних задач, наприклад, обробки ідентичностей, зберігання файлів або створення токенізованих активів. Основною мовою розробки є Rust, яку підтримує середовище Substrate — фреймворк для створення блокчейнів. Polkadot вирішує проблему фрагментації, яка характерна для багатьох блокчейн-систем, об'єднуючи різноманітні рішення в одну екосистему. Водночас система потребує ретельної координації між учасниками для забезпечення сумісності та безпеки, що ускладнює її впровадження.

Кожна з платформ має своє призначення. Ethereum — універсальна, перевірена часом екосистема, яка має найширше коло користувачів і проєктів, але страждає від обмеженої масштабованості. Solana — швидка і дешева, але менш децентралізована і технічно нестабільна. Polkadot — модульна, масштабована та гнучка платформа, яка дозволяє створювати взаємопов'язані, спеціалізовані блокчейни.

З огляду на специфіку, вибір платформи для розробки dApp залежить від потреб проєкту. Якщо важлива стабільність, підтримка та велика спільнота — варто звернути увагу на Ethereum. Якщо пріоритетом є швидкість і низька вартість — Solana стане кращим вибором. Якщо ж мета — створення спеціалізованого блокчейну або мережі взаємодіючих dApps — Polkadot забезпечить необхідну архітектурну гнучкість.

Основні компоненти dApps, які можуть бути вразливими

Децентралізовані додатки (dApps) складаються з декількох взаємозалежних компонентів, кожен з яких може бути потенційно вразливим. Безпека dApps залежить від того, наскільки добре захищені ці компоненти, адже навіть незначна помилка або неправильна конфігурація може призвести до серйозних наслідків, таких як втрата коштів, витік даних або повне порушення функціональності додатка. Основні вразливі компоненти dApps включають:

1. **Смарт-контракти**

- Смарт-контракти є ядром більшості dApps. Це автономні програми, які працюють на блокчейні й управляють логікою бізнес-процесів. Основні вразливості пов'язані з помилками у коді або непередбачуваною поведінкою. Найпоширеніші типи атак:
- Reentrancy attack — зловмисник викликає зовнішній контракт, який знову викликає оригінальний контракт до завершення попереднього виклику, що дозволяє обійти логіку безпеки.

- Integer overflow/underflow — арифметичні помилки, які можуть дозволити змінити значення змінних за межами очікуваного діапазону.
- Необмежений доступ до функцій — помилки у модифікаторах доступу можуть дозволити будь-кому виконувати критичні функції.
- Використання застарілих бібліотек або шаблонів — це створює ризик експлуатації відомих вразливостей.

2. API (інтерфейси прикладного програмування)

Багато dApps використовують API для взаємодії між фронтендом та смарт-контрактами або зовнішніми службами (наприклад, оракулами). Основні ризики:

- Відсутність автентифікації — може дозволити стороннім користувачам здійснювати запити до системи.
- Недостатній контроль доступу — API, який не обмежує, хто і що може викликати, може бути використаний для несанкціонованого доступу або маніпуляцій.
- Відсутність rate limiting — дозволяє атакуючим використовувати методи DDoS для перевантаження API.

3. Фронтенд (веб-інтерфейс користувача)

Фронтенд зазвичай розгортається на централізованих серверах або через IPFS, і може бути ціллю атак незалежно від блокчейна.

- XSS (Cross-Site Scripting) — вставлення зловмисного JavaScript-коду у веб-сторінку.
- Phishing або підміна інтерфейсу — користувача змушують взаємодіяти з підробленим фронтендом, що виглядає як справжній.
- Використання ненадійних бібліотек — зовнішні залежності можуть містити уразливості.

4. Взаємодія з криптогаманцями (MetaMask, Trust Wallet, WalletConnect тощо)

Гаманець — це міст між користувачем і блокчейном. Якщо взаємодія з гаманцем налаштована неправильно, це створює численні загрози:

- Соціальна інженерія — користувача змушують підписати шкідливу транзакцію або дати надмірні дозволи (наприклад, approve() на всі токени).
- Replay-атаки — зловмисник повторно використовує стару транзакцію в іншому контексті.
- Вразливості в Web3-провайдерах — бібліотеки типу web3.js або ethers.js, через які відбувається комунікація з гаманцем, можуть мати власні помилки.

5. Оракули (наприклад, Chainlink)

Оракули — це сервіси, які забезпечують доступ до зовнішніх даних. Хоча вони не є частиною dApp напряму, вони критичні для функціонування багатьох додатків.

- Маніпуляція даними оракула — якщо оракул централізований або недостатньо захищений, зловмисник може надати підроблені дані для впливу на логіку смарт-контракту.
- Залежність від одного джерела — централізований оракул є єдиною точкою відмови (SPOF).

6. Інтеграції з іншими смарт-контрактами або платформами

Багато dApps інтегруються з іншими контрактами або зовнішніми платформами, наприклад, у DeFi-протоколах. Якщо зовнішній контракт буде вразливим або зміниться без попередження, це може негативно вплинути на ваш додаток.

7. Користувацькі дозволи та токен-економіка

Неправильне або надмірне надання дозволів (наприклад, безстрокове approve() на токени ERC-20) може дозволити зловмиснику вивести активи користувача. Також помилки в логіці керування токенами (minting, burning, transfers) можуть призвести до інфляції або крадіжки.

Підсумовуючи, кожен з вищеназваних компонентів відіграє важливу роль у безпеці dApp. Для мінімізації ризиків необхідно застосовувати комплексний підхід до безпеки: аудит смарт-контрактів, аналіз API, регулярне оновлення бібліотек, використання безпечних методів підключення до гаманців, впровадження механізмів багаторівневої автентифікації та уважне тестування взаємодії між усіма компонентами системи.

Аналіз того, як традиційні веб-вразливості (з OWASP) можуть бути застосовні до dApps

Аналіз безпеки децентралізованих додатків (dApps) вимагає врахування як традиційних вразливостей, описаних у списку OWASP Top 10, так і специфічних ризиків, притаманних блокчейн-інфраструктурі. На відміну від централізованих вебзастосунків, dApps поєднують у собі фронтенд, який працює у браузері або мобільному додатку, та бекенд у вигляді смарт-контрактів, що виконуються на публічному або приватному блокчейні. Це створює ширший вектор атак, де можуть одночасно використовуватися як вебвразливості, так і блокчейн-специфічні методи компрометації.

Традиційні вебвразливості (OWASP) в контексті dApps

1. **Injection (SQL/Command Injection)**
   У класичних вебсервісах дозволяє виконання шкідливих команд. У dApps аналогом є **інжекція даних у транзакцію або параметри смарт-контракту**.

Наприклад, якщо смарт-контракт не перевіряє типи вхідних даних або ліміти — це може бути використано для атаки. Крім того, у фронтенді можливі традиційні JavaScript-ін'єкції, якщо не фільтрувати введення користувача.

2. **Broken Authentication**
   Якщо фронтенд dApp не захищає належним чином доступ до функцій (наприклад, не перевіряє, що користувач є власником певного гаманця), можна отримати несанкціонований доступ до функціональності або коштів. Крім того, неправильна логіка в смарт-контракті може дозволити виконувати критичні функції будь-кому.

3. **Sensitive Data Exposure**
   У dApps дані часто зберігаються відкрито на блокчейні. Якщо розробник не зашифрує або не анонімізує чутливу інформацію перед записом, її зможе прочитати будь-хто. Наприклад, записи про транзакції, токени, ідентифікатори користувача можуть видавати особисту інформацію.

4. **XSS (Cross-Site Scripting)**
   Фронтенд dApp піддається XSS так само, як і будь-який вебзастосунок. Якщо користувач може ввести текст, який не буде очищений належним чином, зловмисник може вставити JavaScript-код, який, наприклад, змусить користувача підписати шкідливу транзакцію у своєму гаманці.

5. **Security Misconfiguration**
   Погане налаштування Web3-провайдерів, відсутність HTTPS, відкриті API-інтерфейси або неправильно налаштовані CORS-політики — усе це створює ризики для користувачів dApp. Також погане налаштування смарт-контракту (наприклад, залишені debug-функції) — класичний приклад цієї категорії.

6. **Broken Access Control**
   В dApps це може проявлятися у відсутності обмежень у функціях смарт-контракту (наприклад, будь-хто може викликати mint() або withdraw()). Також у фронтенді відсутність перевірок дозволів може створити критичні прогалини.

7. **Cross-Site Request Forgery (CSRF)**
   Хоча більшість dApps використовує криптографічне підписування транзакцій через гаманці, CSRF може бути використаний для атак на функції не пов’язані з підписанням. Наприклад, фронтенд може мати адміністративну панель або API, які атакуються через CSRF.

Унікальні ризики, пов’язані з блокчейном

1. **Атаки на смарт-контракти**
   На відміну від традиційного бекенду, смарт-контракти не можна змінити після розгортання. Це означає, що помилки залишаються назавжди. Найбільш відомі типи атак:

   - **Reentrancy** (атака на повторний виклик): зловмисник повторно викликає функцію до завершення попереднього виклику.
   - **Integer Overflow/Underflow**: арифметичні переповнення.
   - **Front-running**: атаки, при яких зловмисник бачить незавершену транзакцію і надсилає власну з вищою комісією.
   - **Delegatecall-атаки**: використання зовнішнього коду з власного контексту, що дозволяє змінювати стан контракту неочікувано.

2. **Маніпуляції з оракулами**
   Оракули — критичний компонент для багатьох dApps, які потребують зовнішніх даних (наприклад, ціни активів у DeFi). Якщо джерело даних централізоване або не має перевірки достовірності:

   - Зловмисник може підробити дані.
   - Викликати різке коливання ціни для ліквідації застав.
   - Використовувати **flash loan**-атаки, змінюючи ціни на короткий період.

3. **Проблеми з консенсусом**
   У деяких блокчейнах (особливо з Proof-of-Work) можлива атака 51%, коли зловмисник контролює більшість хешрейту. Це дозволяє йому:

   - Відміняти транзакції (double spending).
   - Цензурувати блоки.
   - Маніпулювати порядком транзакцій.

4. **Sybil-атаки**
   У блокчейн-системах з низькою вартістю створення облікового запису (наприклад, на Solana або EOS) зловмисник може створити тисячі псевдоідентичностей для впливу на голосування або генерацію штучного трафіку.

5. **Gas-атаки та Denial of Service (DoS)**
   Якщо смарт-контракт має погано оптимізований код, зловмисник може створити транзакції з високим газовим споживанням або навмисно змусити контракт “впасти” через перевищення ліміту gas.

6. **Недостатня детермінованість та багатоверсійність контрактів**
   Неправильна реалізація **proxy-контрактів** або **upgradeable patterns** може дозволити змінити логіку контракту з шкідливою метою.

Хоча багато загроз для dApps мають спільне коріння з класичними вебвразливостями (OWASP), екосистема блокчейна додає унікальні виклики: незмінність контрактів, маніпуляції оракулами, взаємодія з гаманцями, консенсусні атаки. Розробники dApps повинні інтегрувати як класичні практики веббезпеки (валідація введення, контроль доступу, очищення виводу), так і спеціалізовані заходи, такі як аудит смарт-контрактів, перевірка оракулів, тестування на поведінку у конкурентному середовищі, та впровадження механізмів оновлення без втрати довіри.

3. Складання вимог безпеки для dApps

Вимоги безпеки для dApps на Ethereum

1. Безпека смарт-контрактів

- **Аудит коду смарт-контрактів**
  Регулярне проведення автоматизованого та ручного аудиту контрактів для виявлення помилок логіки, переповнень чисел, можливих повторних викликів (reentrancy) та інших вразливостей.
  Рекомендації: Використання інструментів типу MythX, Slither, Echidna; залучення зовнішніх аудиторів.
- **Запобігання повторному виклику (Reentrancy attack)**
  Використання патернів типу "checks-effects-interactions" та модифікатора nonReentrant (OpenZeppelin ReentrancyGuard).
- **Обмеження доступу (Access Control)**
  Використання мультипідписних гаманців (multisig) для критичних функцій, а також впровадження ролей (Ownable, Role-Based Access Control).
- **Обробка вхідних даних**
  Перевірка і валідація всіх параметрів, що приймаються смарт-контрактом, аби уникнути неочікуваних станів.

2. Захист фронтенду dApps

- **Захист від XSS (Cross-Site Scripting)**
  Валідація та очищення користувацьких даних перед відображенням у UI; використання Content Security Policy (CSP); уникайте вставки неперевіреного HTML.
- **Безпечна взаємодія з Ethereum Provider**
  Перевірка і явне підтвердження користувачем кожної транзакції через MetaMask або інші гаманці; не зберігати приватні ключі в браузері.

- **Аутентифікація і авторизація**
  Використання Ethereum-підписів для підтвердження прав користувача, зберігання сесій з безпечними токенами.

3. Захист від мережевих атак

- **Шифрування каналів зв'язку**
  Використання HTTPS для всіх API і фронтенд-компонентів.
- **Верифікація контрактів**
  Публічне розміщення коду контрактів у Etherscan для прозорості і довіри.

4. Управління ризиками

- **Моніторинг та логування**
  Відстеження транзакцій і подій контрактів для швидкого виявлення аномалій.
- **Обмеження лімітів транзакцій**
  Встановлення граничних значень для операцій для запобігання зловживань.

Рекомендації щодо захисту від виявлених ризиків

- Проводити регулярні аудити смарт-контрактів перед деплоєм та після значних змін.
- Впроваджувати мультипідписи для управління критичними функціями контрактів.
- Для фронтенду використовувати сучасні практики захисту від XSS: CSP, бібліотеки для безпечного рендерингу (React, Angular), валідацію і санацію вводу.
- Використовувати HTTPS та захищені WebSocket-з'єднання.
- Проводити тестування на проникнення (pentesting) усієї системи dApp.
- Активно моніторити дії користувачів і транзакції в реальному часі, щоб оперативно реагувати на підозрілі операції.

ВИСНОВКИ

У ході виконання дослідження було проаналізовано вимоги OWASP Top 10, які є базовим стандартом для виявлення та класифікації найпоширеніших вразливостей вебдодатків. Ці вимоги є фундаментом для побудови безпечних додатків і залишаються актуальними й у контексті децентралізованих додатків (dApps).

Однак специфіка dApps на базі Ethereum накладає додаткові унікальні виклики, пов'язані зі смарт-контрактами, незмінністю коду, взаємодією з криптогаманцями, маніпуляціями оракулами та ризиками, пов'язаними з консенсусними атаками. Аналіз архітектури та особливостей платформи Ethereum дозволив виділити основні вразливі компоненти dApps: смарт-контракти, API, фронтенд, криптогаманці, оракули та інтеграції з іншими системами.

Поєднання класичних вебвразливостей з особливими ризиками блокчейну підкреслює необхідність комплексного підходу до безпеки. На основі цих досліджень було розроблено адаптовані вимоги безпеки для dApps на Ethereum, які включають регулярний аудит смарт-контрактів, використання мультипідписів для управління критичними функціями, захист фронтенду від XSS-атак, забезпечення безпечної взаємодії з гаманцями та шифрування каналів зв'язку.

Рекомендації щодо управління ризиками включають моніторинг транзакцій, обмеження лімітів операцій, регулярне оновлення та тестування системи. Виконання цих вимог і рекомендацій дозволяє значно знизити ймовірність успішних атак, забезпечити довіру користувачів і стабільність роботи dApp.

Таким чином, ефективна безпека децентралізованих додатків вимагає інтеграції перевірених практик веббезпеки з урахуванням специфіки блокчейну, що є ключовою передумовою для розвитку надійних та масштабованих рішень у сфері децентралізованих технологій.