Доповідач 1
Сьогодні ми розглянемо особливості безпечної розробки та експлуатації децентралізованих додатків (DApps), оскільки їхня архітектура істотно відрізняється від класичних веб-систем. По-перше, DApp працює на основі блокчейн-мережі або іншого peer-to-peer середовища, де кожен вузол (нод) зберігає копію даних або їх частину. Завдяки цьому гарантується стійкість до втрат інформації, а також прозорість усіх транзакцій — жодна дія користувача не може бути прихована або змінена без згоди більшості учасників мережі.

По-друге, логіка в DApp реалізується через смарт-контракти — це невеликі програми, які виконуються безпосередньо в рамках віртуальної машини блокчейну (наприклад, EVM у випадку Ethereum). Вони автоматизують бізнес-процеси й усувають посередника, але водночас стають потенційною точкою виходу коштів чи даних. Навіть одна помилка валідації аргументів або механізмів контролю доступу може призвести до великих фінансових втрат. Окрім типових помилок (переповнення чисел, неправильна обробка вхідних параметрів чи небезпечний виклик зовнішніх контрактів), у децентралізованих системах додатково слід враховувати ризики атаки на консенсус: якщо зловмисник контролює понад 50 % обчислювальних потужностей, він може затримувати чи цензурувати транзакції, а в деяких випадках — змінювати історію (51 % атака).

Третій критичний аспект — безпека приватних ключів. На відміну від звичного пароля, приватний ключ у блокчейні одночасно є підписом і доступом до всіх активів. Якщо ключ потрапляє до рук зловмисника, подальші наслідки можуть бути незворотними. Крім того, DApps часто інтегруються з додатковими сервісами: ораклами (Chainlink), децентралізованим зберіганням (IPFS, Swarm) тощо. Якщо ці шари не захищені належним чином (наприклад, через неправильні схеми реплікації чи слабку авторизацію), виникає ризик DoS-атак, маніпуляції даними або витоку конфіденційної інформації.

Наша мета — не тільки ознайомитися з внутрішньою будовою DApp, а й оглянути відомі зразки вразливості OWASP Top 10 для класичних веб-додатків.

Доповідач 2
Давайте коротко розкажу OWASP Top 10 та продемонструємо, як кожен із пунктів стосується традиційних веб-додатків:

1. **A1: Неправильний контроль доступу (Broken Access Control)**
   • Неправильне налаштування ролей і прав, що дозволяє неавторизованому користувачу читати або змінювати чужі дані.
2. **A2: Нестача криптографічного захисту (Cryptographic Failures)**
   • Слабкі алгоритми хешування, незашифроване зберігання паролів, неправильна реалізація TLS.
3. **A3: Ін’єкції (Injection)**
   • SQL-ін’єкції, XSS, NoSQL-ін’єкції, коли дані користувача «впливають» на запит без валідації.
4. **A4: Небезпечний дизайн (Insecure Design)**
   • Відсутність чіткого threat modeling, невраховані сценарії зловмисних дій на етапі проєктування.
5. **A5: Неправильна конфігурація (Security Misconfiguration)**
   • Відкриті директорії, неправильні заголовки безпеки, дефолтні облікові записи.
6. **A6: Уразливі та застарілі компоненти (Vulnerable and Outdated Components)**
   • Використання застарілих версій фреймворків або бібліотек із відомими CVE.

7. **A7: Проблеми ідентифікації та аутентифікації (Identification and Authentication Failures)**
   • Слабкі токени, неправильне керування сесіями, недостатні перевірки під час входу.
8. **A8: Відсутність перевірки цілісності (Software and Data Integrity Failures)**
   • Можливість підміни JavaScript-файлів через CDN, відсутність перевірки підписів оновлень.
9. **A9: Проблеми логування та моніторингу (Security Logging and Monitoring Failures)**
   • Відсутність логів безпеки, ускладнене виявлення інцидентів у реальному часі.
10. **A10: SSRF (Server-Side Request Forgery)**
    • Коли сервер виконує запити до внутрішніх або приватних ресурсів за ініціативою користувача.

Ми протестували приклади A3 (XSS, SQLi) за допомогою OWASP ZAP, а також вивчили безпечне зберігання секретів A2 (bcrypt, KDF, KMS-сервіси). Водночас звернули увагу на A5 (налаштування веб-серверів) та A9 (централізовані логи з SIEM).

Тепер розглянемо, як ці ризики проявляються у DApp:

- **Injection → Reentrancy, Integer Underflow/Overflow:**
  • Коли смарт-контракт дозволяє «переривати» (reenter) функцію під час виконання, зловмисник може вивести кошти кілька разів.
  • Недостатня перевірка арифметичних операцій призводить до некоректних результатів.
- **Authentication Failures → Безпека приватного ключа та мульти-підпис:**
  • Зберігання приватного ключа в незашифрованому кошику (keystore) або відсутність двухфакторної аутентифікації.
  • У традиційних веб-сервісах ми маємо login/password, а тут — підпис транзакції = аутентифікація, тож компрометація ключа = повний контроль.
- **Data Integrity Failures → Проксі-контракти та механізм upgradability:**
  • Під час оновлення контракту можуть змінитись його функції, що може дати змогу зловмиснику внести нову вразливу логіку.

Аналогічні вимоги безпеки для DApps

На основі аналізу OWASP Top 10, адаптуємо ключові положення для середовища Ethereum DApp:

№	OWASP-категорія	Вимога для DApp

1	**Broken Access Control**	**DBC1: Smart-Contract Access Control**— Використовувати модифікатори `onlyOwner`, `roles` (OpenZeppelin AccessControl).— Виклики чутливих функцій мають проходити через мультисиг (Gnosis Safe) або Timelock.
2	**Cryptographic Failures**	**DBC2: Secure Key & Crypto Management**— Не зберігати приватні ключі у фронтенді; використовувати апаратні гаманці (Ledger, Trezor) або зовнішні KMS.— RPC-зв'язок повинен бути тільки через TLS (HTTPS).
3	**Injection**	**DBC3: Reentrancy та Integer Safety**— Застосовувати `ReentrancyGuard` (OpenZeppelin) та бібліотеку `SafeMath`/вбудовані перевірки Solidity ^0.8.x.— Уникати небезпечних викликів `call`/`delegatecall` без ретельної валідації.
4	**Insecure Design**	**DBC4: Threat Modeling & Modular Architecture**— Перед розробкою контрактів провести аналіз загроз (STRIDE/DREAD).— Розбивати функціонал на модульні контракти (Token, Governance, Treasury) для зручного аудиту.
5	**Security Misconfiguration**	**DBC5: Harden RPC & Node Settings**— Обмежити CORS для RPC-нод, не використовувати публічні endpoint без фільтрації.— Перевіряти налаштування нод (firewall, версії клієнта, плагіни).
6	**Vulnerable and Outdated Components**	**DBC6: Up-to-Date Dependencies**— Регулярно оновлювати OpenZeppelin Contracts, Hardhat, Truffle, Ethers.js/Web3.js.— Виконувати `npm audit`/`yarn audit` і застосовувати патчі.
7	**Identification and Authentication Failures**	**DBC7: Off-Chain Auth & Nonce Management**— Для Sign-In with Ethereum використовувати nonce із бекенду;— Перевіряти `chainId` (EIP-155) у підписаних транзакціях, щоб уникнути replay-атак.
8	**Software and Data Integrity Failures**	**DBC8: Contract Verification & Frontend SRI**— Публікувати код контракту для верифікації на Etherscan;— Використовувати Subresource Integrity (SRI) для скриптів, якщо фронтенд розгортається через CDN/IPFS.
9	**Security Logging and Monitoring Failures**	**DBC9: On-Chain Events & Monitoring**— Кожен метод, що змінює стан, має emit Event;— Інтегруватися з Tenderly, TheGraph або Dune для моніторингу наявності аномалій.
10	**Server-Side Request Forgery (SSRF)**	**DBC10: Secure Oracle & Proxy**— Для ораклів обмежувати список дозволених URL;— Для всієї взаємодії з RPC використовувати власний проксі (тільки потрібні JSON-RPC методи).

Додаткова таблиця: Порівняння типових загроз у веб-додатку та їхніх аналогів у DApp

Традиційна загроза	Приклад у Web App	Аналогічна загроза у DApp	Контрзаходи (DApp)
SQL-ін'єкція	Вставка «SELECT * FROM users WHERE id = ?»	Виклик невалідівних	Перевірка ABI-кодованих даних; уникати

		`delegatecall/call` з байтами	динамічного виклику функцій
XSS	Ін’єкція JS у форму від користувача	Зловмисний Smart-Contract Metadata (JSON, URI)	Екраування полів у Frontend; не використовувати innerHTML
Атака CSRF	Форсування відправки форми з чужого сайту	Фейковий DApp, який запрошує підпис транзакції	Показувати деталі транзакції; підписувати тільки те, що згенеровано бекендом
Переповнення буфера	Ін’єкція коду через функцію, що записує у фіксовану пам'ять	Integer Overflow/Underflow у Solidity < 0.8	Використовувати Checked Math у Solidity ≥ 0.8; застосовувати SafeMath
Вразливий Session ID	Брак обмежень HTTP-Only/Cookie; сесійні токени крадуться	Відкритий Private Key у LocalStorage або незахищений JSON Keystore	Зберігати лише адреси; ключі — у MetaMask або апаратних гаманцях
SSRF	Запити з сервера на внутрішні ресурси (metadata)	Oracle Bridge виконує запити до внутрішніх API	Whitelisting URL у Oracle; API-Proxy із фільтрацією методів
Використання старих бібліотек	jQuery 1.x з відомими CVE	OpenZeppelin v2 без ReentrancyGuard	Оновити до OpenZeppelin v4; налаштувати CI для перевірки залежностей
Відсутність логування	Немає запису спроб входу, пожежного логу	Смарт-контракт не emits Event; немає off-chain ведення	Emit усіх критичних подій; збирати логи у зовнішню систему SIEM

Висновок:

1. Класичні вразливості OWASP Top 10 мають свої специфічні аналоги в середовищі DApps (Reentrancy ↔ Injection, ключі ↔ паролі тощо).
2. Для кожного ризику необхідно визначити власні контрзаходи: перевірка ролей у контрактах, безпечне зберігання приватних ключів, аудити смарт-контрактів, моніторинг on-chain подій.

3. Таблиці вище дають змогу швидко зорієнтуватися у критичних точках: що було в Web-світі та як це перетворюється у блокчейн-середовищі.

Дякуємо за увагу, готові відповісти на запитання!