Лабораторна робота №3

Дослідження безпеки децентралізованих блокчейн-систем та адаптація вимог OWASP

Мета роботи: дослідити специфічні загрози безпеки децентралізованих блокчейн-систем та розробити адаптовані вимоги безпеки на основі методології OWASP

Завдання:

1. Проаналізувати архітектурні особливості децентралізованих блокчейн-систем
2. Ідентифікувати унікальні векти атак, властиві блокчейн-інфраструктурі
3. Розробити Blockchain Security Framework (BSF-10) на основі принципів OWASP

Частина 1: Архітектура децентралізованих блокчейн-систем

1.1 Компоненти децентралізованої системи

Децентралізована блокчейн-система складається з наступних рівнів:

Рівень консенсусу (Consensus Layer)

- **Механізми консенсусу:** Proof of Work (PoW), Proof of Stake (PoS), Delegated PoS, Byzantine Fault Tolerance
- **Валідатори/майнери:** Учасники, що підтверджують транзакції
- **Блок-продюсери:** Вузли, що створюють нові блоки

Мережевий рівень (Network Layer)

- **P2P мережа:** Розподілена мережа вузлів без центрального сервера
- **Протоколи синхронізації:** Gossip protocol, Block propagation
- **Топологія мережі:** Full nodes, Light nodes, Archive nodes

Рівень даних (Data Layer)

- **Блокчейн структура:** Ланцюжок блоків з криптографічними хешами
- **Merkle Trees:** Ефективна верифікація транзакцій
- **State Storage:** Зберігання поточного стану системи

Прикладний рівень (Application Layer)

- **Смарт-контракти:** Програми, що виконуються на блокчейні
- **DApps:** Децентралізовані додатки з Web3 інтерфейсом
- **Протоколи взаємодії:** Web3.js, Ethers.js, JSON-RPC

1.2 Унікальні характеристики блокчейн-систем

Immutability (Незмінність)

- Дані, записані в блокчейн, неможливо змінити або видалити
- Помилки в смарт-контрактах стають постійними
- Необхідність механізмів оновлення через proxy-контракти

Transparency (Прозорість)

- Всі транзакції та дані публічні
- Вихідний код смарт-контрактів доступний для аналізу
- Можливість аналізу поведінки через block explorers

Decentralization (Децентралізація)

- Відсутність єдиної точки контролю
- Управління через консенсус учасників
- Стійкість до цензури та відмов

Pseudonymity (Псевдонімність)

- Користувачі ідентифікуються адресами, а не іменами
- Можливість аналізу транзакцій для деанонімізації
- Privacy coins для покращеної конфіденційності

Частина 2: Blockchain Security Framework (BSF-10)

BSF-1: Консенсус-рівень атаки (Consensus Layer Attacks)

Опис: Атаки, спрямовані на компрометацію механізму консенсусу блокчейн-мережі.

51% атака (Majority Attack)

- **Сценарій:** Контроль >50% hashrate (PoW) або stake (PoS)
- **Наслідки:** Можливість reorg атак, подвійних витрат, цензури
- **Реальні випадки:** Ethereum Classic (2019), Bitcoin SV (2021)

Приклад розрахунку атаки на PoW:

```
Вартість атаки = (Hashrate мережі × 0.51) × Вартість обладнання
Bitcoin: ~150 EH/s × 51% ≈ $15+ млрд
```

Nothing-at-Stake (PoS специфічна)

- **Проблема:** Валідатори можуть підтримувати кілька форків одночасно
- **Рішення:** Slashing умови, finality гаджети

Eclipse атака

- **Мета:** Ізоляція вузла від решти мережі
- **Метод:** Контроль всіх з'єднань жертви через Sybil вузли
- **Захист:** Diverse peer selection, outbound connections

Засоби захисту:

- Збільшення децентралізації мережі
- Checkpointing механізми
- Slashing умови для PoS
- Різноманітність клієнтів (client diversity)

BSF-2: Смарт-контракт вразливості (Smart Contract Vulnerabilities)

Опис: Специфічні вразливості програмного коду, що виконується на блокчейні.

Reentrancy атаки

Механізм: Рекурсивний виклик функції до завершення попереднього виконання.

solidity

```solidity
// ВРАЗЛИВИЙ КОД
contract VulnerableBank {
    mapping(address => uint256) public balances;

    function withdraw() public {
        require(balances[msg.sender] > 0);

        // Зовнішній виклик ПЕРЕД зміною стану
        (bool success,) = msg.sender.call{value: balances[msg.sender]}("");
        require(success);

        balances[msg.sender] = 0; // ЗАНАДТО ПІЗНО!
    }
}

// АТАКУЮЧИЙ КОНТРАКТ
contract Attacker {
    VulnerableBank bank;

    function attack() public payable {
        bank.deposit{value: msg.value}();
        bank.withdraw(); // Початок атаки
    }

    receive() external payable {
        if (address(bank).balance > 0) {
            bank.withdraw(); // Рекурсивний виклик!
        }
    }
}
```

Захист через ReentrancyGuard:

```solidity
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract SecureBank is ReentrancyGuard {
    mapping(address => uint256) public balances;

    function withdraw() public nonReentrant {
        require(balances[msg.sender] > 0);

        uint256 amount = balances[msg.sender];
        balances[msg.sender] = 0; // State змінюємо ПЕРШИМ

        (bool success,) = msg.sender.call{value: amount}("");
        require(success);
    }
}
```

Integer Overflow/Underflow

До Solidity 0.8.0:

```solidity
// ВРАЗЛИВІСТЬ
uint256 balance = 0;
balance = balance - 1; // Underflow: 0 - 1 = 2^256 - 1
```

Сучасний захист (Solidity >=0.8.0):

```solidity
// Автоматичні перевірки overflow/underflow
uint256 balance = 0;
balance = balance - 1; // REVERT з помилкою
```

Delegatecall небезпеки

```solidity
// НЕБЕЗПЕЧНО: delegatecall зберігає контекст викликаючого контракту
contract Proxy {
    address public implementation;

    function upgrade(address newImpl) public {
        // Відсутня перевірка власника!
        implementation = newImpl;
    }

    fallback() external payable {
        implementation.delegatecall(msg.data); // Може змінити storage!
    }
}
```

BSF-3: Криптографічні вразливості (Cryptographic Vulnerabilities)

Слабка випадковість

Проблема: Використання передбачуваних джерел ентропії.

```solidity
// НЕБЕЗПЕЧНО
function random() public view returns (uint256) {
    return uint256(keccak256(
        abi.encodePacked(block.timestamp, block.difficulty, msg.sender)
    )) % 100; // Майнер може маніпулювати!
}

// БЕЗПЕЧНО через Chainlink VRF
import "@chainlink/contracts/src/v0.8/VRFConsumerBase.sol";

contract SecureRandom is VRFConsumerBase {
    bytes32 internal keyHash;
    uint256 internal fee;

    function getRandomNumber() public returns (bytes32 requestId) {
        require(LINK.balanceOf(address(this)) >= fee);
        return requestRandomness(keyHash, fee);
    }
}
```

Signature replay атаки

```solidity
// ВРАЗЛИВО: відсутність nonce
function transfer(address to, uint256 amount, bytes memory signature) public {
    bytes32 hash = keccak256(abi.encodePacked(to, amount));
    address signer = recoverSigner(hash, signature);
    require(signer == owner);

    // Підпис можна використати повторно!
    token.transfer(to, amount);
}

// ЗАХИСТ: додати nonce та chainId
mapping(address => uint256) public nonces;

function secureTransfer(
    address to,
    uint256 amount,
    uint256 nonce,
    bytes memory signature
) public {
    require(nonce == nonces[msg.sender]);

    bytes32 hash = keccak256(abi.encodePacked(
        to, amount, nonce, block.chainid, address(this)
    ));

    address signer = recoverSigner(hash, signature);
    require(signer == msg.sender);

    nonces[msg.sender]++;
    token.transfer(to, amount);
}
```

BSF-4: Економічні атаки (Economic Attack Vectors)

Flash Loan атаки

Механізм: Використання миттєвих позик для маніпуляції ринком.

Типовий сценарій Flash Loan атаки:

1. Взяти flash loan 10M USDC з Aave
2. Обміняти USDC на Token A в DEX #1 → ціна Token A зростає
3. Використати завищену ціну Token A як колатерал в протоколі кредитування
4. Взяти максимальну позику проти завищеного колатералу
5. Повернути flash loan + комісію
6. Залишити прибуток від експлойту

Реальний приклад - bZx атака (2020):

```
1. Flash loan: 7,500 ETH з dYdX
2. Обмін 5,500 ETH → DAI на Kyber (ціна ETH/DAI зросла)
3. Короткий продаж ETH на bZx за завищеною ціною
4. Повернення flash loan
5. Прибуток: ~$350,000
```

Захист від Flash Loan атак:

- Time-Weighted Average Price (TWAP) оракули
- Multiple price feeds з різних джерел
- Circuit breakers при різких змінах ціни
- Limits на операції в одній транзакції

MEV (Maximal Extractable Value)

Типи MEV:

- **Front-running:** Розміщення транзакції перед жертвою
- **Back-running:** Розміщення після для арбітражу
- **Sandwich attacks:** Оточення жертви з двох сторін

Приклад Sandwich атаки:

```
Mempool стан:
1. Користувач купує 100 ETH за USDC (великий ордер)
2. MEV бот детектує транзакцію

MEV бот виконує:
1. Front-run: Купити ETH → ціна зростає
2. Дозволити виконатися транзакції користувача → ціна зростає ще більше
3. Back-run: Продати ETH за вищою ціною → прибуток
```

BSF-5: Оракул маніпуляції (Oracle Manipulation)

Проблема: Залежність від зовнішніх джерел даних створює векти атак.

Price Oracle атаки

```solidity
// ВРАЗЛИВО: використання одного джерела ціни
contract VulnerableLending {
    function borrow(uint256 collateralAmount) public {
        uint256 price = priceOracle.getPrice("ETH/USD");
        uint256 maxBorrow = (collateralAmount * price * 75) / 100;
        // Якщо оракул скомпромітований - катастрофа!
    }
}

// БЕЗПЕЧНО: агрегація кількох джерел
contract SecureLending {
    IPriceOracle[] public oracles;
    uint256 public constant MAX_PRICE_DEVIATION = 500; // 5%

    function getSecurePrice(string memory pair) public view returns (uint256) {
        uint256[] memory prices = new uint256[](oracles.length);

        for (uint i = 0; i < oracles.length; i++) {
            prices[i] = oracles[i].getPrice(pair);
        }

        uint256 median = calculateMedian(prices);

        // Перевірка на аномальні відхилення
        for (uint i = 0; i < prices.length; i++) {
            uint256 deviation = abs(prices[i] - median) * 10000 / median;
            require(deviation <= MAX_PRICE_DEVIATION, "Price deviation too high");
        }

        return median;
    }
}
```

BSF-6: Міжблокчейнові мости (Cross-Chain Bridge Vulnerabilities)

Архітектура мостів:

- **Lock & Mint:** Заморожування токенів на Chain A, випуск на Chain B
- **Burn & Mint:** Спалювання на Chain A, випуск на Chain B
- **Liquidity Pools:** Пули ліквідності на обох мережах

Типові вразливості мостів:

Signature Verification Failures:

```solidity
// ВРАЗЛИВА ВЕРИФІКАЦІЯ
function verifyWithdraw(
    uint256 amount,
    address recipient,
    bytes[] memory signatures
) public {
    bytes32 hash = keccak256(abi.encodePacked(amount, recipient));

    uint256 validSigs = 0;
    for (uint i = 0; i < signatures.length; i++) {
        address signer = recoverSigner(hash, signatures[i]);
        if (isValidator[signer]) {
            validSigs++;
        }
    }

    // ПРОБЛЕМА: не перевіряється унікальність підписантів!
    require(validSigs >= requiredSigs);
    mint(recipient, amount);
}
```

Ronin Bridge експлойт (2022, $600M):

- 5 з 9 validator ключів скомпромітовано
- Атакуючі створили фіктивні withdrawal транзакції
- Недостатня верифікація підписів дозволила витягнути 173,600 ETH

BSF-7: Governance атаки (Governance Attacks)

Flash Loan Governance атаки

Сценарій:

1. Взяти flash loan governance токенів
2. Проголосувати за шкідливий proposal (наприклад, зміна параметрів протоколу)
3. Виконати proposal в тій же транзакції
4. Повернути токени

Реальний випадок - Beanstalk (2022, $180M):

```solidity
// Атакуючий використав flash loan для отримання >50% voting power
// Прийняв proposal для відправки коштів на свій адрес
// Все в одній транзакції
```

Захист:

- Timelock для всіх governance рішень
- Voting delay periods
- Minimum voting periods
- Quadratic voting або інші механізми проти plutocracy

BSF-8: Layer 2 специфічні ризики (Layer 2 Specific Risks)

Sequencer централізація

- Більшість L2 мають централізований sequencer
- Single point of failure та censorship
- Потенційна MEV extraction на L2 рівні

State commitment проблеми

```solidity
// Optimistic Rollup - fraud proof вікно
// Користувачі мають 7 днів для challenge неправильних state transitions
// Якщо challenge не надійде - state вважається валідним
```

Data availability ризики

- Якщо L2 sequencer не публікує дані - користувачі не можуть відновити стан
- Потрібні гарантії data availability

BSF-9: Privacy і traceability (Privacy Attacks)

Blockchain аналітика

- **Cluster Analysis:** Групування адрес одного користувача
- **Transaction Graph Analysis:** Відстеження потоків коштів
- **Timing Analysis:** Корелювання транзакцій за часом

Приклад деанонімізації:

```python
# Псевдокод аналізу Bitcoin транзакцій
def cluster_addresses(transactions):
    clusters = {}
    for tx in transactions:
        # Heuristic: всі inputs належать одному користувачеві
        input_addresses = [inp.address for inp in tx.inputs]
        merge_clusters(clusters, input_addresses)

    return clusters
```

MEV-boost та private mempools

- Flashbots Protect для захисту від front-running
- Private transaction pools
- Compromise між privacy та transparency

BSF-10: Node і Infrastructure атаки (Infrastructure Attacks)

Eclipse атаки на повні вузли

```python
# Атакуючий контролює всі peer connections жертви
victim_connections = [
    "malicious_node_1:8333",
    "malicious_node_2:8333",
    "malicious_node_3:8333",
    # ... всі 8-10 connections
]

# Жертва отримує тільки фіктивні дані від атакуючого
# Може призвести до прийняття неправильних транзакцій
```

RPC endpoint атаки

- **Rate limiting bypass:** Використання множинних IP
- **State bloat attacks:** Запити, що споживають багато ресурсів
- **MEV extraction:** Front-running через privileged RPC access

Захист infrastructure:

- Load balancing між multiple RPC providers
- Rate limiting та authentication
- DDoS protection
- Geographical distribution of nodes

Частина 3: Інструменти та методології аудиту

3.1 Статичний аналіз смарт-контрактів

Slither - найпопулярніший інструмент

```bash
# Установка
pip install slither-analyzer

# Основне використання
slither contract.sol

# Конкретні детектори
slither contract.sol --detect reentrancy-eth,uninitialized-state
```

Slither детектори за категоріями BSF:

```yaml
BSF-2 (Smart Contract):
  - reentrancy-eth: Reentrancy vulnerabilities
  - uninitialized-state: Uninitialized state variables
  - tx-origin: Dangerous use of tx.origin

BSF-3 (Cryptographic):
  - weak-prng: Weak pseudo-random number generation
  - encode-packed-collision: Hash collision with encodePacked

BSF-5 (Oracle):
  - price-oracle: Price oracle manipulation risks
```

Mythril - символічне виконання

```bash
# Docker запуск
docker run -v $(pwd):/tmp mythril/myth analyze /tmp/contract.sol

# Специфічні модулі
myth analyze contract.sol -m delegatecall,external_calls
```

3.2 Динамічне тестування та фузинг

Echidna - property-based тестування

```solidity
// echidna_test.sol
contract EchidnaTest {
    Bank bank;

    constructor() {
        bank = new Bank();
    }

    // Invariant: баланс завжди >= сума депозитів
    function echidna_balance_invariant() public returns (bool) {
        return address(bank).balance >= bank.totalDeposits();
    }

    // Property: withdraw не може вивести більше депозиту
    function test_withdraw_limit(uint256 amount) public {
        bank.deposit{value: amount}();
        uint256 balanceBefore = address(this).balance;
        bank.withdraw();
        assert(address(this).balance - balanceBefore <= amount);
    }
}
```

```bash
# Запуск Echidna
echidna-test contract.sol --contract EchidnaTest
```

Foundry фузинг

solidity

```solidity
// test/FuzzTest.t.sol
contract FuzzTest is Test {
    function testFuzzWithdraw(uint256 depositAmount) public {
        vm.assume(depositAmount > 0 && depositAmount < 1e20);

        bank.deposit{value: depositAmount}();
        uint256 balanceBefore = address(this).balance;

        bank.withdraw();

        assertLe(
            address(this).balance - balanceBefore,
            depositAmount,
            "Withdrew more than deposited"
        );
    }
}
```

3.3 Моніторинг в реальному часі

Forta Network - децентралізований моніторинг

javascript

```javascript
// Forta agent для детекції flash loan атак
const {
  Finding,
  FindingSeverity,
  FindingType
} = require("forta-agent");

async function handleTransaction(txEvent) {
  const findings = [];

  // Детекція flash loan в одній транзакції
  const flashLoanEvents = txEvent.filterLog(FLASH_LOAN_ABI);
  const largeTransfers = txEvent.filterLog(TRANSFER_ABI, {
    topics: [],
    valueThreshold: ethers.utils.parseEther("1000")
  });

  if (flashLoanEvents.length > 0 && largeTransfers.length > 5) {
    findings.push(Finding.fromObject({
      name: "Potential Flash Loan Attack",
      description: "Multiple large transfers with flash loan in single tx",
      alertId: "FLASH-LOAN-ATTACK",
      severity: FindingSeverity.High,
      type: FindingType.Suspicious,
      metadata: {
        flashLoanAmount: flashLoanEvents[0].args.amount.toString(),
        transferCount: largeTransfers.length.toString()
      }
    }));
  }

  return findings;
}
```

Частина 4: Практичне завдання - Аудит DeFi протоколу

4.1 Опис тестового протоколу

Розглянемо спрощений DEX протокол `SimpleSwap`, що імплементує базовий AMM (Automated Market Maker):

solidity

```solidity
// УВАГА: Код містить навмисні вразливості для навчальних цілей!
pragma solidity ^0.8.0;

contract SimpleSwap {
    mapping(address => mapping(address => uint256)) public liquidity;
    mapping(address => uint256) public totalSupply;

    event Swap(address indexed user, address tokenIn, address tokenOut, uint256 amountIn, uint2
    event AddLiquidity(address indexed user, address tokenA, address tokenB, uint256 amountA, u

    // ВРАЗЛИВІСТЬ BSF-2: Відсутність reentrancy protection
    function swap(
        address tokenIn,
        address tokenOut,
        uint256 amountIn
    ) external {
        require(amountIn > 0, "Amount must be positive");

        uint256 reserveIn = IERC20(tokenIn).balanceOf(address(this));
        uint256 reserveOut = IERC20(tokenOut).balanceOf(address(this));

        // ВРАЗЛИВІСТЬ BSF-4: Простий розрахунок без захисту від маніпуляцій
        uint256 amountOut = (amountIn * reserveOut) / (reserveIn + amountIn);

        // ВРАЗЛИВІСТЬ BSF-2: Зовнішній виклик ПЕРЕД зміною стану
        IERC20(tokenOut).transfer(msg.sender, amountOut);
        IERC20(tokenIn).transferFrom(msg.sender, address(this), amountIn);

        emit Swap(msg.sender, tokenIn, tokenOut, amountIn, amountOut);
    }

    // ВРАЗЛИВІСТЬ BSF-5: Використання поточного балансу як оракула ціни
    function getPrice(address tokenA, address tokenB) external view returns (uint256) {
        uint256 reserveA = IERC20(tokenA).balanceOf(address(this));
        uint256 reserveB = IERC20(tokenB).balanceOf(address(this));
        return (reserveA * 1e18) / reserveB;
    }
}
```

4.2 Покрокове виявлення вразливостей

Крок 1: Статичний аналіз з Slither

bash

```bash
slither SimpleSwap.sol --detect reentrancy-eth,external-function
```

Очікувані результати:

```
SimpleSwap.swap() (lines 15-28) allows reentrancy:
    - External call: IERC20(tokenOut).transfer(msg.sender, amountOut) (line 25)
    - State variables written after the call: None (but logic can be exploited)
```

Крок 2: Аналіз економічної логіки

Проблема з формулою AMM:

```solidity
// Поточна формула: amountOut = (amountIn * reserveOut) / (reserveIn + amountIn)
// Правильна формула з комісією:
uint256 amountInWithFee = amountIn * 997; // 0.3% комісія
uint256 numerator = amountInWithFee * reserveOut;
uint256 denominator = (reserveIn * 1000) + amountInWithFee;
uint256 amountOut = numerator / denominator;
```

Крок 3: Тестування reentrancy атаки

solidity

```solidity
contract ReentrancyAttacker {
    SimpleSwap public target;
    IERC20 public tokenA;
    IERC20 public tokenB;
    bool private attacking;

    constructor(address _target, address _tokenA, address _tokenB) {
        target = SimpleSwap(_target);
        tokenA = IERC20(_tokenA);
        tokenB = IERC20(_tokenB);
    }

    function attack() external {
        tokenA.approve(address(target), 1000 ether);
        attacking = true;
        target.swap(address(tokenA), address(tokenB), 100 ether);
    }

    // Malicious contract receive function
    function onERC20Received(
        address,
        address,
        uint256,
        bytes calldata
    ) external returns (bytes4) {
        if (attacking && tokenB.balanceOf(address(target)) > 0) {
            attacking = false; // Prevent infinite recursion
            target.swap(address(tokenB), address(tokenA), 50 ether);
        }
        return this.onERC20Received.selector;
    }
}
```

4.3 Виправлення вразливостей

solidity

```solidity
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract SecureSwap is ReentrancyGuard, Pausable, Ownable {
    mapping(address => mapping(address => uint256)) public liquidity;
    mapping(address => uint256) public totalSupply;

    uint256 public constant FEE_DENOMINATOR = 10000;
    uint256 public swapFee = 30; // 0.3%

    // BSF-5: TWAP оракул для захисту від маніпуляцій
    struct PriceAccumulator {
        uint256 price0CumulativeLast;
        uint256 price1CumulativeLast;
        uint32 blockTimestampLast;
    }

    mapping(bytes32 => PriceAccumulator) public priceAccumulators;

    modifier validTokens(address tokenA, address tokenB) {
        require(tokenA != tokenB, "Identical tokens");
        require(tokenA != address(0) && tokenB != address(0), "Zero address");
        _;
    }

    // BSF-2: Захист від reentrancy + правильний порядок операцій
    function swap(
        address tokenIn,
        address tokenOut,
        uint256 amountIn,
        uint256 minAmountOut
    ) external nonReentrant whenNotPaused validTokens(tokenIn, tokenOut) {
        require(amountIn > 0, "Amount must be positive");

        // Checks - отримуємо поточні резерви
        uint256 reserveIn = IERC20(tokenIn).balanceOf(address(this));
        uint256 reserveOut = IERC20(tokenOut).balanceOf(address(this));
        require(reserveIn > 0 && reserveOut > 0, "Insufficient liquidity");

        // BSF-4: Правильна формула AMM з комісією
        uint256 amountInWithFee = amountIn * (FEE_DENOMINATOR - swapFee);
        uint256 numerator = amountInWithFee * reserveOut;
        uint256 denominator = (reserveIn * FEE_DENOMINATOR) + amountInWithFee;
        uint256 amountOut = numerator / denominator;
```

```
    require(amountOut >= minAmountOut, "Insufficient output amount");
    require(amountOut < reserveOut, "Insufficient liquidity");

    // Effects - змінюємо стан ПЕРЕД зовнішніми викликами
    _updatePriceAccumulator(tokenIn, tokenOut, reserveIn, reserveOut);

    // Interactions - зовнішні виклики в кінці
    IERC20(tokenIn).transferFrom(msg.sender, address(this), amountIn);
    IERC20(tokenOut).transfer(msg.sender, amountOut);

    emit Swap(msg.sender, tokenIn, tokenOut, amountIn, amountOut);
}

// BSF-5: TWAP оракул замість spot price
function getTWAP(
    address tokenA,
    address tokenB,
    uint256 timeWindow
) external view returns (uint256) {
    bytes32 pairId = keccak256(abi.encodePacked(tokenA, tokenB));
    PriceAccumulator memory accumulator = priceAccumulators[pairId];

    require(block.timestamp > accumulator.blockTimestampLast + timeWindow,
            "Time window too short");

    uint256 timeElapsed = block.timestamp - accumulator.blockTimestampLast;
    uint256 avgPrice = accumulator.price0CumulativeLast / timeElapsed;

    return avgPrice;
}

function _updatePriceAccumulator(
    address tokenA,
    address tokenB,
    uint256 reserveA,
    uint256 reserveB
) internal {
    bytes32 pairId = keccak256(abi.encodePacked(tokenA, tokenB));
    PriceAccumulator storage accumulator = priceAccumulators[pairId];

    uint32 blockTimestamp = uint32(block.timestamp % 2**32);
    uint32 timeElapsed = blockTimestamp - accumulator.blockTimestampLast;

    if (timeElapsed > 0 && reserveA != 0 && reserveB != 0) {
        accumulator.price0CumulativeLast +=
            uint256(UQ112x112.encode(reserveB).uqdiv(reserveA)) * timeElapsed;
        accumulator.price1CumulativeLast +=
            uint256(UQ112x112.encode(reserveA).uqdiv(reserveB)) * timeElapsed;
    }
```

```
        accumulator.blockTimestampLast = blockTimestamp;
    }

    // BSF-10: Emergency functions
    function pause() external onlyOwner {
        _pause();
    }

    function unpause() external onlyOwner {
        _unpause();
    }

    // BSF-4: Circuit breaker для захисту від великих торгів
    uint256 public maxSwapAmount = 1000000 ether;

    modifier swapLimits(uint256 amount) {
        require(amount <= maxSwapAmount, "Swap amount too large");
        _;
    }
}
```

Частина 5: Кейс-стаді реальних атак

5.1 The DAO атака (2016) - BSF-2

Контракт: The DAO на Ethereum

Збиток: 3.6M ETH (~$50M на той час)

Тип атаки: Reentrancy

Уразливий код:

```solidity
function splitDAO(
    uint _proposalID,
    address _newCurator
) noEther onlyTokenholders returns (bool _success) {
    // ... перевірки ...

    // ПРОБЛЕМА: зовнішній виклик перед зміною балансу
    if (balances[msg.sender] > 0) {
        // Створення дочірнього DAO з балансом користувача
        DAO newDAO = createNewDAO(_newCurator);

        // Переказ ефіру НА НОВИЙ КОНТРАКТ
        if (newDAO.call.value(balances[msg.sender])()) {
            // Тільки тепер обнулення балансу - ЗАНАДТО ПІЗНО!
            balances[msg.sender] = 0;
            return true;
        }
    }
}
```

Атака:

1. Атакуючий створив зловмисний контракт, що реалізував fallback функцію
2. При отриманні ефіру fallback викликав `splitDAO` знову
3. Рекурсивно виводив ефір до вичерпання контракту

Наслідки: Ethereum Hard Fork на Ethereum і Ethereum Classic

5.2 Wormhole атака (2022) - BSF-6

Протокол: Wormhole Bridge (Solana ↔ Ethereum)
Збиток: $320M
Тип атаки: Signature verification bypass

Проблема в коді:

```rust
// Спрощений код верифікації підписів
fn verify_signatures(
    vaa: &MessageData,
    signatures: &[Signature]
) -> Result<(), Error> {
    let mut last_guardian_index = None;
    let mut valid_signatures = 0;

    for sig in signatures {
        // ПРОБЛЕМА: недостатня перевірка guardian index
        if let Some(last_index) = last_guardian_index {
            if sig.guardian_index <= last_index {
                continue; // Skip duplicate, але не fail!
            }
        }

        if verify_signature(&vaa.hash, sig) {
            valid_signatures += 1;
        }

        last_guardian_index = Some(sig.guardian_index);
    }

    if valid_signatures >= REQUIRED_SIGNATURES {
        Ok(())
    } else {
        Err(Error::InsufficientSignatures)
    }
}
```

Експлойт:

1. Атакуючий знайшов застарілий guardian set в системі
2. Створив VAA (Verified Action Approval) з цим guardian set
3. Обійшов перевірку актуальності через логічну помилку
4. Mint 120,000 wETH на Ethereum без backing активів

5.3 Euler Finance атака (2023) - BSF-4

Протокол: Euler Finance lending protocol
Збиток: $197M
Тип атаки: Flash loan + liquidation manipulation

Схема атаки:

```solidity
// Кроки атаки в одній транзакції:
1. flash_loan(30M DAI) from Aave
2. deposit(20M DAI) into Euler → отримати eDAI
3. mint(200M eDAI) - збільшити debt позицію
4. donate(10M DAI) до резерву → збільшити exchange rate
5. liquidate власну позицію з вигодою через маніпульований rate
6. repeat кроки 4-5 кілька разів
7. withdraw прибуток
8. repay flash loan
```

Проблема: Donating до резерву миттєво змінювало exchange rate, що дозволяло маніпулювати liquidation logic.

Частина 6: Підсумок та рекомендації

6.1 Blockchain Security Maturity Model

Рівень 1: Базова безпека

- [ ] Використання перевірених шаблонів (OpenZeppelin)
- [ ] Статичний аналіз всіх контрактів
- [ ] Unit tests з coverage >80%
- [ ] Basic access controls

Рівень 2: Проміжна безпека

- [ ] Formal verification критичних функцій
- [ ] Property-based тестування (Echidna)
- [ ] Time locks для governance
- [ ] Multi-signature для admin функцій
- [ ] Bug bounty програма

Рівень 3: Продвинута безпека

- [ ] Математичне моделювання економічної безпеки
- [ ] Real-time моніторинг (Forta agents)
- [ ] Circuit breakers та emergency pausing
- [ ] Comprehensive incident response план
- [ ] Regular security audits (кожні 6 місяців)

Рівень 4: Професійна безпека

- [ ] Zero-knowledge proof системи для privacy
- [ ] MEV protection механізми
- [ ] Decentralized security infrastructure
- [ ] Automated formal verification pipeline
- [ ] Security-focused governance model

6.2 Інструменти для кожного BSF рівня

yaml

```yaml
BSF-1 (Consensus Attacks):
  Detection:
    - Chainlink monitoring services
    - Custom validators watchers
    - Network hashrate/stake monitoring
  Prevention:
    - Client diversity initiatives
    - Slashing conditions implementation
    - Economic security analysis

BSF-2 (Smart Contract):
  Static Analysis: [Slither, Mythril, Manticore, Securify]
  Dynamic Testing: [Echidna, Foundry, Hardhat]
  Formal Verification: [Certora, KEVM, Dafny]

BSF-3 (Cryptographic):
  Tools: [Cryptol, Tamarin Prover, EasyCrypt]
  Best Practices: [Use audited libraries, Never roll own crypto]

BSF-4 (Economic Attacks):
  Simulation: [Gauntlet, Block Analitica]
  Monitoring: [Forta agents, Custom MEV detection]
  Protection: [Flashbots Protect, Private mempools]

BSF-5 (Oracle Manipulation):
  Solutions: [Chainlink Price Feeds, Band Protocol, Witnet]
  Monitoring: [Price deviation alerts, Volume spike detection]

BSF-6 (Cross-Chain):
  Audit Focus: [Multi-sig verification, State proof validation]
  Monitoring: [Bridge balance tracking, Unusual withdrawal alerts]

BSF-7 (Governance):
  Protection: [Timelock contracts, Multi-sig DAOs, Quadratic voting]
  Monitoring: [Proposal tracking, Voting power concentration]

BSF-8 (Layer 2):
  Audit: [Sequencer decentralization, Data availability proofs]
  Monitoring: [L2 state root verification, Withdrawal delays]

BSF-9 (Privacy):
  Tools: [Chainalysis, Elliptic, OXT]
  Protection: [Mixers, ZK-SNARKs, Private pools]

BSF-10 (Infrastructure):
  Protection: [Load balancers, DDoS protection, Geographic distribution]
  Monitoring: [Node health checks, Network partitioning detection]
```

6.3 Checklist для production deployment

Pre-deployment (4-6 тижнів)

- [ ] **Архітектурний огляд** - threat modeling, attack surface analysis
- [ ] **Код-ревю** - manual review, automated scanning
- [ ] **Тестування** - unit, integration, property-based tests
- [ ] **Аудит** - мінімум 2 незалежні аудиторські фірми
- [ ] **Формальна верифікація** - критичних інваріантів

Deployment процес (1-2 тижні)

- [ ] **Testnet deployment** - повне тестування на testnet >2 тижнів
- [ ] **Bug bounty** - публічна програма з винагородою
- [ ] **Документація** - повна технічна документація
- [ ] **Моніторинг setup** - алерти, дашборди, incident response
- [ ] **Emergency procedures** - pause mechanisms, upgrade procedures

Post-deployment (ongoing)

- [ ] **Continuous monitoring** - 24/7 спостереження
- [ ] **Regular audits** - повторні аудити кожні 6 місяців
- [ ] **Community engagement** - відкритий діалог з користувачами
- [ ] **Security updates** - оновлення згідно з новими загрозами

6.4 Майбутні виклики безпеки

Quantum Computing загрози

- **Shor's algorithm** - загроза для RSA та ECDSA
- **Grover's algorithm** - зменшення ефективності хеш-функцій
- **Потрібна міграція** на quantum-resistant криптографію

AI/ML загрози

- **Automated exploit generation** - AI пошук вразливостей
- **Social engineering** - deepfake атаки на governance
- **MEV optimization** - AI для максимізації MEV extraction

Regulatory compliance

- **AML/KYC вимоги** для DeFi протоколів
- **GDPR** та право на забуття в immutable системах
- **Securities regulation** для governance токенів

Висновки

Дослідження показало, що традиційні підходи до веб-безпеки потребують кардинальної адаптації для децентралізованих блокчейн-систем. Розроблений **Blockchain Security Framework (BSF-10)** відображає унікальні загрози, властиві децентралізованій архітектурі.

Ключові відмінності від традиційної веб-безпеки:

1. **Immutability** - помилки стають постійними
2. **Economic incentives** - нові типи атак через фінансову мотивацію
3. **Transparency** - публічність коду створює нові векти атак
4. **Consensus dependency** - загрози на рівні мережевого консенсусу
5. **Cross-chain complexity** - ризики міжблокчейнової взаємодії

Пріоритетні напрямки розвитку безпеки:

- **Формальна верифікація** смарт-контрактів
- **Економічне моделювання** стійкості протоколів
- **Автоматизований моніторинг** в реальному часі
- **Quantum-resistant** криптографічні рішення
- **Cross-chain security** стандарти

Безпека децентралізованих систем вимагає мультидисциплінарного підходу, що поєднує криптографію, економіку, теорію ігор та розподілені системи.