МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

НАДОЗІРНИЙ Святослав Вікторович

Алгоритми та системи захисту смарт-контрактів на основі блокчейну / Algorithms and protection systems of blockchain based smart contracts

спеціальність: 125 – Кібербезпека

освітньо-професійна програма –Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
С.В.Надозірний
_____

Науковий керівник
д.т.н., професор В.В.Яцків
_____

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2022 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ - 2022

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 - Кібербезпека
освітньо-професійна програма –Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
____" _____ 2021 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
НАДОЗІРНИЙ Святослав Вікторович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми та системи захисту смарт-контрактів на основі блокчейну / Algorithms and protection systems of blockchain based smart contracts
керівник роботи д.т.н., професор В.В. Яцків

затверджені наказом по університету від 31 грудня 2021 року № 606

2. Строк подання студентом закінченої випускної кваліфікаційної роботи 16 листопада 2022 року.

3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно дослідити:
- дослідити структуру токенів ERC-20 та ERC-721;
- дослідити основні джерела загроз;
- дослідити існуючі інструменти аналізу смарт контрактів;
- спроектувати вимоги до аудиту смарт контактів;
- описати загальний підхід для мінімізації загроз в процесі впровадження смарт контрактів в блокчейн.

5. Перелік графічного матеріалу у роботі.
- Будова токенів.
- Графічна діаграма структури смарт контракту.

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 11 жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз типових токенів	12.2021 р. – 03.2022 р.	
2	Джерела загроз і утиліти аналізу	03.2022 р. – 05.2022 р.	
3	Проведення аудиту смарт-контракту	05.2022 р. – 11.2022 р.	

Студент _____ Надозірний С.В.
(підпис)

Керівник роботи _____ д.т.н., професор В.В. Яцків
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми та системи захисту смарт-контрактів на основі блокчейну» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека» освітньо-професійної програми «Кібербезпека» написана обсягом 73 сторінки і містить 2 ілюстрації, 1 таблицю, 8 додатків та 24 джерела за переліком посилань.

Метою кваліфікаційної роботи є підвищення ефективності захисту смарт контрактів завдяки уніфікації методології аудитів.

У роботі проведено дослідження структури токенів ERC-20 та ERC-721, проведено аналіз основних вразливостей, перераховано існуючі інструменти для аналізу смарт контрактів. Визначено основні вимоги аудиту до смарт контактів і сформовано загальний підхід для мінімізації загроз в процесі впровадження смарт контрактів в блокчейн.

Результати роботи можуть успішно застосовуватися для аудиту смарт контрактів основі блокчейн.

Ключові слова: БЛОКЧЕЙН, СМАРТ КОНТРАКТИ, БЕЗПЕКА СМАРТ КОНТРАКТІВ, АТАКИ НА БЛОКЧЕЙН, АУДИТ СМАРТ КОНТРАКТІВ.

ANNOTATION

The qualification work on "Algorithms and protection systems of blockchain based smart contracts" for the Master's degree in the specialty 125 "Cybersecurity" of the educational and professional program "Cybersecurity" is written in the volume of 73 pages and contains 2 illustrations, 1 table, 8 appendices and 24 sources according to the list of links.

The purpose of the qualification work is to increase the efficiency of smart contract protection due to the unification of the audit methodology.

The paper examines the structure of ERC-20 and ERC-721 tokens, analyzes the main vulnerabilities, and lists the existing tools for analyzing smart contracts. The main audit requirements for smart contacts have been determined and a general approach has been formed to minimize threats in the process of implementing smart contracts in the blockchain.

The results of the work can be successfully applied to the audits of blockchain-based smart contracts.

Keywords: BLOCKCHAIN, SMART CONTRACTS, SECURITY OF SMART CONTRACTS, ATTACKS ON BLOCKCHAIN, AUDIT OF SMART CONTRACTS.

ЗМІСТ

ВСТУП

Актуальність розвитку блокчейн технологій підтверджують статистичні дані, впродовж року, за інформацією представленою на сайті https://coinmarketcap.com/. Цифрова революція сприяла новим видам встановлення відносин та контрактів. Прообразом смарт-контрактів стали паперові угоди, які існують у будь-якій діяльності сучасних організацій. Після складання такі контракти зазвичай вручну підписуються і опечатуються, після чого підписанти виконують озвучені зобов'язання.[1]

В 1996 році, американський фахівець з криптографії Нік Сабо вперше запропонував концепцію «смарт-контрактів». Він стверджував, що:

«Нові інституції і нові способи формалізації відносин цих інституцій стали можливі завдяки цифровій революції. Я називаю ці контракти «розумними» тому що вони більш функціональні, ніж їхні неживі паперові предки. Не передбачається використання штучного інтелекту. Смарт-контракти – це набір обіцянок у цифровому форматі, включно з протоколами, за якими сторони виконують ці обіцянки». [2]

Сабо описав смарт-контракт як комп'ютерний протокол, який на основі математичних алгоритмів самостійно проводить операції з повним контролем за їх виконанням. Таке визначення смарт-контрактів актуально до сих пір. Після появи таких криптовалют як Bitcoin, впровадили спеціальні реєстри, де майном є гроші. Bitcoin і багато похідних протоколів містять механізми для того, щоб реалізувати можливості спільної власності і виконання контрактів. Віталіком Бутеріним в 2015 була заснована платформа для реалізації смарт контрактів – Ethereum, яка користується найбільшою популярністю, оскільки має ряд переваг. На відміну від Bitcoin, смарт-контракти Ethereum розробляються на одній з мов, спроектованих для трансляції в байт-код віртуальної машини Ethereum – Solidity, Vyper і Serpent, LLL, Mutan. [3][4]

Сторони підписують розумний контракт, використовуючи методи, аналогічні підписанню перерахунку коштів в криптовалютні мережі, що

існують в наш час. Після підписання сторонами, контракт зберігається в блокчейні і вступає в силу. Для забезпечення автоматизованого виконання зобов'язань договору обов'язково потрібно середовище існування (ноди блокчейна Ethereum), яке дозволяє повністю автоматизувати виконання пунктів контракту. Це означає, що розумні контракти можуть існувати тільки всередині середовища, що має доступ байт-коду контракту і сховища даних блокчейну.

Але як і будь-яка комп'ютерна програма, смарт контракти можуть мати вразливості і можуть бути атаковані. Останні популярні атаки на смарт контракти:[5][6][7][8]

− Uniswap/Lendf.Me (квітень 2020) – 25 млн.$, використано reentrancy-атаку.

− BurgerSwap (травень 2021) – 7.2 млн.$, фейковий токен і використання reentrancy exploit.

− SURGEBNB (Серпень 2021) – 4 млн.$, reentrancy-based price manipulation attack.

− Cream finance (серпень 2021) – 18.8 млн.$, вразливість reentrancy дозволила робити повторні запозичення.

− Siren protocol hack (вересень 2021) – 3.5 млн.$, використання reentrancy attack

− Cream finance (жовтень 2021) - 130 млн.$, flash loan to perform price manipulation

− MonoX (листопад 2021) - 31 млн.$, - price override vulnerability

Мета роботи. Дослідження алгоритмів роботи смарт-контрактів і розробка систем захисту смарт-контрактів.

Об’єкт дослідження – процеси розробки сумісних з блокчейн Ethereum смарт-контрактів.

Предмет дослідження – токени на базі смарт контрактів в таких блокчейнах – Ethereum, BinanceSmartCoin, Polygon, Tronx та інші сумісні з Ethereum блокчейни.

В рамках дослідження ставляться такі завдання:

- дослідити структуру токенів ERC-20 та ERC-721;
- дослідити основні джерела загроз;
- дослідити існуючі інструменти аналізу смарт контрактів;
- спроектувати вимоги до аудиту смарт контактів;
- описати загальний підхід для мінімізації загроз в процесі впровадження смарт контрактів в блокчейн.

Методи дослідження. Теоретичною базою дослідження є сучасні теорії і наукові концепції, монографії та періодичні публікації зарубіжних вчених з питань інформаційних технологій, криптографії. У роботі використано готові розробки і вихідний код з відкритих джерел.

Інформаційну базу дослідження склали:

1. Вихідні коди смарт-контрактів https://etherscan.io
2. Вихідні коди смарт-контрактів https://bscscan.io
3. Проект https://github.com/OpenZeppelin/ethernaut
4. Проект https://github.com/Uniswap/v2-core
5. Проект https://github.com/pancakeswap/pancake-smart-contracts

1 АНАЛІЗ СТРУКТУРИ ТОКЕНІВ

1.1. Основи роботи смарт-контрактів в блокчейні

Нижче показано представлено візуалізовану схему, загалом для будь-якої децентралізованої системи криптовалюти зі смарт-контрактами, як приклад для Ethereum:

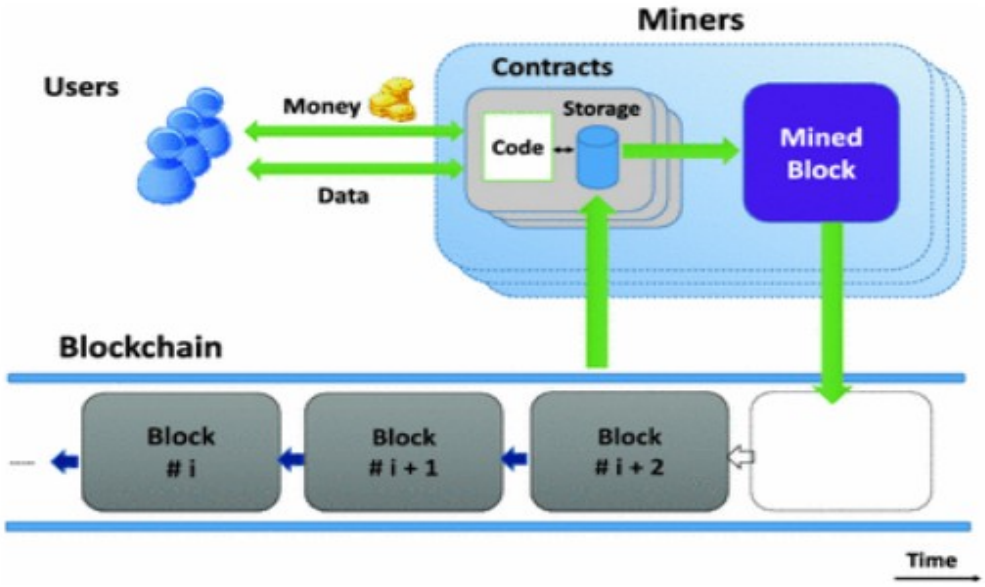


Рисунок 1.1 – Будова токенів

Код, який зберігається в блокчейні після розгортання, є низькорівневим байт-кодом на основі стека (код EVM), що представляє мову програмування високого рівня, на якій спочатку написані смарт-контракти. В результаті можна сказати, що оскільки байт-код є загальнодоступним із блокчейну, поведінка смарт-контрактів цілком передбачувана, і його код може бути перевірений кожним вузлом мережі.

Самим популярним компілятором байткоду для EVM є Solidity. Solidity – статично типізована JavaScript-подібна мова програмування, створена для розробки розумних контрактів, які працюють на віртуальній машині Ethereum (EVM). Програми на мові Solidity транслюються в байткод EVM. Solidity дозволяє розробникам створювати самодостатні програми, що містять бізнес-логіку, результуючу в транзакційні записи блокчейну.

Використання синтаксису ECMAScript за задумом Вуда має допомогти прийняттю мови дійсними веброзробниками. Однак, на відміну від ECMAScript, мова отримала статичну типізацію змінних і динамічні типи значень. Порівняно з компільованими в такий же байт-код мовами Serpent і Mutan, мова Solidity має важливі відмінності. Підтримуються комплексні змінні контрактів, включаючи довільні ієрархічні відображення (mappings) і структури. Контракти підтримують спадкування, включаючи множинне і C3-лінеаризацію. Підтримується бінарний інтерфейс програмування (ABI), що має безліч типобезпечних функцій в кожному контракті (пізніше з'явився також і у Serpent). Специфікована система документування коду, для пояснення послідовності викликів, що отримала назву «Специфікації природною мовою Ethereum» (Ethereum Natural Format Specification).

Приклад програми на мові Solidity:

```
contract UAH
{
  mapping(address=>uint) balances;
  uint constant totalAmount = 100000000000;

  /// Mint 100 billion UAH tokens for creator.
  constructor(uint256 amount){
      balances[msg.sender] = amount;
  }

  /// Send `amount` UAH tokens from the account of `sender`, to an
account accessible only by `to`.
  function send(address to, uint256 amount) {
    if (balances[msg.sender] >= amount) {
      balances[to] += amount;
      balances[msg.sender] -= amount;
    }
  }

  /// getter function for the balance
  function balance(address owner) constant returns (uint256 balance) {
    balance = balances[owner];
  }

}
```

Крім того, смарт-контракти можуть зберігати стан, обмінюватися цифровими активами, приймати введення користувачів, зберігати дані, отримувати інформацію від зовнішніх служб і виражати бізнес-логіку. Після розгортання коду смарт-контракту його функції запускаються повідомленнями та/або транзакціями, надісланими на адресу смарт-контракту. Смарт-контракт завжди повинен бути детермінованим, тобто той самий вхід завжди буде давати однаковий результат. Інакше, враховуючи, що кожне завдання виконується на кожному вузлі мережі, це створить проблемний стан при досягненні консенсусу всередині вузлів, а отже, відхилить весь процес.

Оскільки ми представляємо смарт-контракти як дуже перспективні програми, у цьому розділі будуть розглянуті основи можливих випадків використання смарт контрактів та їх категоризація. Існує кілька спроб категоризувати сфери застосування смарт-контрактів.

Початкова категоризація виконана Бутеріном (засновником Ethereum) на основі трьох верхніх рівнів:

Фінансові програми:

- Субвалюти;
- похідні фінансові інструменти;
- контракти на хеджування;
- ощадні гаманці або повномасштабні трудові контракти.

Квазіфінансові програми:

- самозабезпечення винагород;
- хмарні обчислення;
- азартні ігри.

Нефінансові програми:

- онлайн-голосування;
- децентралізоване управління;
- ринки прогнозів.

1.2. Токени ERC-20

ERC (Ethereum Request for Comments) – назва офіційного протоколу для внесення пропозицій щодо покращення мережі Ethereum.

Токени ERC-20 є одними з найпопулярніших. Це цифрові активи, розроблені, випущені і використовуються так само, як біткоін, за винятком того, що вони працюють виключно на блокчейні Ethereum або блокчейнах сумісних з EVM. Ці токени використовують певний смарт-контракт, який відстежує транзакції цього токена.

Біткоїн задав парадигму для інших криптопроектів: щоб випустити якусь цифрову валюту, потрібно спочатку запустити окремий блокчейн. Ethereum зламав це правило. За допомогою смарт-контрактів будь-який розробник отримав можливість випустити свій токен і надати унікальні корисні функції в рамках свого додатка.

Проте, до появи ERC-20 існувала проблема сумісності між різними токенами, адже кожен із них мав унікальний смарт-контракт. Інакше кажучи, щоб біржа чи гаманець могли підтримувати токен, його творцям щоразу потрібно було писати зовсім новий код.

Таким чином, підтримка зростаючої кількості токенів ставала все більш проблематичною, займаючи занадто багато часу. Для вирішення цієї проблеми було створено стандартний протокол для всіх токенів.

ERC-20 значно спростив та уніфікував випуск токенів у рамках однієї мережі. Цей стандарт поряд зі смарт-контрактами став універсальним способом створення та монетизації додатків, які використовують блокчейн. Так Ethereum став першою в історії блокчейн-платформою.

За даними Etherscan, в основній мережі Ethereum станом на кінець 2022 року існує понад 600 000 ERC-20-сумісних токенів. У тому числі безліч топових цифрових валют з ринковою капіталізацією, які служать різним цілям:[10]

- Tether (USDT);

- Maker (MKR);
- Chainlink (LINK);
- The Sandbox (SAND);
- The Graph (GRT);
- Uniswap (UNI);
- Axie Infinity (AXS);
- Aave (AAVE);
- ApeCoin (APE);
- Basic Attention Token (BAT);
- Compound (COMP);
- OMG Network (OMG);
- yearn.finance (YFI);
- 1inch (1INCH);
- Enjin Coin (ENJ).

ERC-20 популярні не просто так, і ось кілька факторів, які роблять їх цікавими:

- Зручність. Токени ERC-20 прості та легкі у використанні. Тому, що Ethereum смарт-контракти пишуться мовою програмування Solidity. Він схожий на JavaScript. Крім того, розробники можуть також кодувати смарт-контракти за допомогою мови програмування Vyper, схожої на Python.
- Гнучка настройка. Залежно від бізнес-логіки та взаємодії користувачів токени ERC-20 можна налаштовувати. Можливо включити такі функції, як автоматичне поповнення газу для майбутніх транзакцій, заморожування та розморожування токена, додавання до центрального монетного двору для зміни токенів у обігу та багато іншого.
- План розробки. Стандарт ERC-20 дає розробникам ясний план, який дозволяє створювати нові токени без особливих зусиль.
- Стандартизація токенів. Ethereum надає характеристики токена, які включають правила взаємодії між різними токенами та правила купівлі

токенів. За допомогою універсального стандарту користувачі можуть переводити нові токени на гаманець і одразу виставляти їх на біржу.

− Ліквідність. Якщо проекти, засновані на Ethereum, активні та взаємодіють один з одним, це залучається більше проектів та користувачів до мережі Ethereum. Існує також таке рішення, як Uniswap, конвертація токенів ERC-20 з ним стала ще простішою.

− Популярність та поширеність. ERC-20, як і його токени, впізнавані на більшості бірж та гаманців. Це відбувається головним чином через універсальний протокол, який може бути адаптований для різноманітних обмінів. Крім того, його взаємозамінність робить його ідеальним для торгових додатків.

− Зниження рівня шахрайства. Усі транзакції мають бути схвалені, а загальна кількість токенів полегшує процес перевірки, це гарантує відсутність дублікатів токенів у зверненні.

Недоліки токенів на базі ERC-20:

− Повільні транзакції. Висновок та переклад працюють у блокчейні ефіру. Коли мережа перевантажена, всі передачі на ERC-20 будуть уповільнені. Хоча очікується, що оновлення у 2023 році «Шардинг» вирішить цю проблему.

− Незворотні транзакції. Немає ніякого способу повернути кошти, якщо користувачі відправили токени ERC-20 не за тією адресою, і назавжди залишаються в контрактах. Те саме стосується і токенів, які були вкрадені зловмисниками, одним із найяскравіших прикладів є злом DAO.

− Дуже низький рівень входу. Критики кажуть, що людям надто легко створювати свої токени без певної мети. В результаті розробники можуть легко використовувати цю політику для розробки шахрайських ICO та токенів з проектами, які не становлять цінності.

Основні характеристики протоколу ERC-20.

Цей стандарт передбачає шість обов'язкових та три опціональні (але рекомендовані) параметри для будь-якого смарт-контракту.

Серед обов'язкових параметрів:

- функція **totalSupply**, яка відповідає за загальну емісію токенів, забезпечуючи неможливість створення нових токенів після досягнення максимального числа.
- **balance0f** визначає початкову кількість токенів, приписаних до певної адреси. Зазвичай це адреса, що належить емітенту.

Також стандарт визначає два методи переміщення токенів.

- Функція **transfer** забезпечує передачу токенів користувачам
- **transferFrom** необхідна для використання трансферу в контрактах.

Ще дві функції потрібні для верифікації двох попередніх методів переміщення токенів:

- **approve** служить для перевірки того, що смарт-контракт, виходячи із загальної емісії, може здійснювати дистрибуцію токенів;
- **allowance** необхідна для перевірки наявності достатнього балансу для надсилання токенів на іншу адресу.

Серед необов'язкових параметрів - назва токена **name** та його код **symbol**, а також визначення максимальної кількості дробових цифр після коми **decimals** (наприклад, WETH має вісімнадцять таких цифр - 1.000000000000000000 USDT).

Набір цих нескладних у реалізації параметрів дозволяє вести єдину кодову базу, яка взаємодіє з будь-яким смарт-контрактом ERC-20. Типовий інтерфейс для токену ERC-20:

```
abstract contract IERC20 {
    event Transfer(address indexed _from, address indexed _to, uint256
_value)
     event Approval(address indexed _owner, address indexed _spender,
uint256 _value)
  function name() public view returns (string)
  function symbol() public view returns (string)
  function decimals() public view returns (uint8)
  function totalSupply() public view returns (uint256)
  function balanceOf(address _owner) public view returns (uint256
  balance)
```

```
  function transfer(address _to, uint256 _value) public returns (bool
  success)
  function transferFrom(address _from, address _to, uint256 _value)
  public returns (bool success)
  function approve(address _spender, uint256 _value) public returns
  (bool success)
  function allowance(address _owner, address _spender) public view
  returns (uint256 remaining)
}
```

1.3. Токени ERC-721

NFT (абревіатура від англійської «non-fungible token») — це невзаємозамінні токени, які є правом власності на різні цифрові об'єкти: тексти, зображення, аудіозаписи, цифрові твори мистецтва, ігрові предмети або персонажі, доменні імена, фінансові інструменти, клубні карти і т.д.

Якщо криптовалюти умовно взаємозамінні — наприклад, один біткоїн у гаманці одного користувача дорівнює і тотожний одному біткоїну в гаманці іншого користувача, — то один NFT-токен, що представляє картину, не дорівнює одному NFT-токену іншого користувача, оскільки це можуть бути різні картини різних художників із різною вартістю.

Кожен з NFT-токенів неповторний і існує в однині. Незамінні токени унікальні - їх не можна скопіювати. Кожен містить ідентифікуючу інформацію, записану в смарт-контрактах. Ця інформація робить кожен NFT відмінним від іншого.

За допомогою NFT розробники вирішили проблему забезпечення прав власності на цифрові об'єкти. Інформацію про власника та його токені закріплюють у блокчейні. Замінити або видалити інформацію неможливо.

Першим найпопулярнішим в історії NFT стали CryptoKitties. Запущений у листопаді 2017 року проект дає можливість розводити котиків. Користувач бере двох NFT-котиків і виробляє від них NFT-нащадок різного ступеня рідкості, яке залишає собі або продає.

Багато проектів почали експериментувати з механікою розведення NFT, додаючи інші ігрові елементи.

ERC-721 - перша та найпопулярніша реалізація NFT на Ethereum, що розширює можливості базового стандарту ERC-20. У ній кожен тип токенів потребує окремого смарт-контракту.

NFT-проекти розробляють в основному в мережах Ethereum, Flow та WAX, а також на базі рішень другого рівня та сайдчейнів.

Платформи для випуску та маркетплейси можна розділити на такі категорії:

- Агрегатори, що забезпечують купівлю та продаж NFT.
- Універсальні протоколи випуску, які забезпечують створення NFT.
- Нішеві маркетплейси: переважають маркетплейси або емітенти творів мистецтва, але поступово виникають NFT-блоги, такі, як Mirror, і маркетплейси у сфері музики – наприклад, EulerBeats.

NFT-агрегатори

- Open Sea

NFT-маркетплейси/протоколи випуску арт-об'єктів

- SuperRare
- Async Art
- Known Origin
- Nifty Gateway
- Foundation
- Portion
- MakersPlace

Універсальні маркетплейси/протоколи випуску

- Rarible
- Zora
- Mintbase

Нішеві протоколи випуску контенту

- Mirror

- Audius
- EulerBeats.

Мета Всесвіт - наступне покоління цифрових платформ. Включає цілий стек технологій, у тому числі віртуальну та доповнену реальності, блокчейн та штучний інтелект. Це віртуальна "пісочниця", в якій простір створюють самі користувачі. Вони ж самостійно обирають чим займатися. За допомогою криптовалют та NFT у метавсесвітах можлива монетизація будь-якої діяльності.

Компанія Meta, яка раніше називалася Facebook, розвиває власний метавсесвіт Horizon Worlds. Марк Цукерберг описав метавсесвіт як «інтернет, в якому ви перебуваєте, а не просто дивіться на нього».

У масовому уявленні метавсесвіти - це нове покоління інтернет-платформ: самодостатні світи, користувачі яких зможуть у них працювати, відпочивати, навчатися та здійснювати будь-які інші справи. Кожен із таких світів може мати свої функції та особливості.

Концепція метавсесвітів передбачає, що її користувачів існуватимуть необмежені можливості для монетизації свого віртуального життя.
Головний елемент, який лежить в основі економіки віртуального світу - невзаємозамінні токени (NFT). Вони є правами власності на будь-які цифрові предмети та майно, які створюють користувачі.

Крупні компанії все активніше беруть участь у метавсесвітах. Так, навесні 2022 року в Decentraland пройшов тиждень моди, в якому взяли участь багато відомих брендів, включаючи Dolce & Gabbana та Tommy Hilfiger. А провідний виробник косметики Estee Lauder рекламував нові продукти.[12]

Платіжна компанія Mastercard подала заявки на 15 торгових марок, пов'язаних із метавсесвітами та криптовалютами. McDonald's реєструє патент, згідно з яким мережа планує відкрити ресторани у метавсесвітах та продавати в них їжу у формі NFT.[11]

Корпорації вкладають великі суми створення метавсесвітів. Зокрема, розробник ігор Epic Games залучив відразу $2 млрд на створення розважальних віртуальних світів «для всієї родини». Він реалізує проект у співпраці з Sony та LEGO.[13]

А в Південній Кореї метавсесвіт зробили національним проектом і виділили на його розробку $186 млн із державного бюджету.

Різні організації відкривають свої представництва у віртуальних світах: наприклад, такі офіси є у банку JPMorgan та крипторегулятора Дубая.

За оцінками Citibank, до 2030 року розмір ринку метавсесвітів сягне $13 трлн, а кількість користувачів — 5 млрд осіб.

Типовий інтерфейс для токену ERC-721:

```
abstract contract IERC721 {
    event Transfer(address indexed _from, address indexed _to, uint256
    indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved,
    uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed
    _operator, bool _approved);
    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256
    _tokenId, bytes data) external payable;
    function safeTransferFrom(address _from, address _to, uint256
    _tokenId) external payable;
    function transferFrom(address _from, address _to, uint256 _tokenId)
    external payable;
    function approve(address _approved, uint256 _tokenId) external
    payable;
    function setApprovalForAll(address _operator, bool _approved)
    external;
    function getApproved(uint256 _tokenId) external view returns
    (address);
    function isApprovedForAll(address _owner, address _operator)
    external view returns (bool);
}
```

1.4. Типові загрози та їх експлуатація

Виклик функції в смарт-контракті є формою транзакції. Виклик може створити транзакцію з частиною даних, що містить підпис функції та аргументи. Абонент може надіслати ETH разом із транзакцією. Коли транзакція фіксується в блоці та розподіляється між вузлами в мережі, кожна EVM викликає функцію з аргументами і повертає результат виконання. Ефір, надісланий разом із транзакцією, буде додано до балансу рахунку смарт-контракту.

Оскільки кожен може прочитати код і побачити результати, властивість прозорості надається смарт-контрактам і процесу їх виконання. Код також може бути виконаний кожною машиною, щоб забезпечити послідовність результатів виконання. Хоча зберігання більшості блокчейн-систем і код смарт-контракту є незмінними, а стан змінних у коді смарт-контракту – ні. Зокрема, такі змінні можуть бути змінені шляхом виконання логіки програмування коду. Якщо код має деякі вразливості, будь-хто зі зловмисними намірами може маніпулювати кодом для своєї вигоди або для порушення функцій смарт-контракту. Таким чином, вкрай важливо визначити вразливі місця смарт-контракту, щоб код можна було належним чином захистити.

Є багато випадків, викликаних недоліками програмування смарт-контрактів.

Незважаючи на те, що є багато робіт для виявлення вразливостей смарт-контрактів, досі немає досліджень щодо вивчення поточного стану вразливих смарт-контрактів. Дослідження має на меті виявити загальні випадки та тенденції вразливостей у смарт-контрактах, а також визначити загальні характеристики вразливих смарт-контрактів.

Повторний вхід (Re-entrancy). У Solidity є три функції, які використовуються для переказу деякої валюти на зовнішню адресу; вони надсилають (transfer), передають (send) і викликають (call). Однак у випадку, якщо адресою призначення є смарт-контракт, ці функції також діють як виклик функції для «резервної функції» в смарт-контракті призначення (receive). Зловмисний контракт може використати цей факт для створення «створеної запасної функції» для виконання чогось у вихідному контракті.

```
contract Victim{
   mapping(address => uint) wallets;
  function deposit() public payable{
       wallets[msg.sender] = msg.value;
  }
  function withdraw(uint amount) public{
     require(wallets[msg.sender] >=amount);
     msg.sender.send(amount);
     wallets[msg.sender] -= amount
  }
}

contract Attacker{
    address _victim = 0x0123456789abcdef;
    function attack() public payable{
       Victim(_victim).deposit().value(10);
       Victim(_victim).withdraw(10);
    }
    receive() public payable{
       Victim(_victim).withdraw(10);
    }
}
```

Це приклад атаки повторного входу на функцію withdraw, і він схожий на причину інциденту, який стався у випадку DAO (Siegel, 2016). Зловмисник починає зі створення власного зловмисного смарт-контракту (Attacker) і викликає функцію відкликання смарт-контракту жертви. Після проходження деякої перевірки на першому рядку контракт надсилає Ether до смарт-контракту зловмисника. Оскільки призначенням переказу є смарт-контракт, виконується резервна функція receive, яка, у свою чергу, знову викликає функцію withdraw. Оскільки баланс зловмисника ще не вираховано, смарт-контракт Victim знову надішле ефір. Цикл виконання продовжується, доки баланс смарт-контракту жертви не стане нульовим або поки не вичерпається газ транзакції.

Цілочисельне переповнення (Integer overflow). Цілочисельне переповнення є поширеними проблемами, які також зустрічаються в інших мовах програмування. Цілочисельне переповнення відбувається під час збільшення значення змінної обмеженого розміру, доки воно не перевищить максимальну ємність. Потім значення змінної переходить до найменшого значення, як показано в наступному фрагменті коду:

```
uint8 users=255;
users +=1;
```

Цей код створює змінну num_users як 8-розрядне ціле число без знаку, що має діапазон можливих значень від 0 до 255, і ініціює її значення до 255. Коли змінна збільшується на 1 і не може зберігати значення 256, значення скорочується. лише 8 біт і замість них зберігаються нулі. Той самий принцип застосовується для цілочисельного недоповнення.

Залежність від позначки часу (Timestamp Dependency). Оскільки смарт-контракт працює на віртуальній машині Ethereum (EVM), яка надає лише інформацію про сам смарт-контракт, тобто його транзакції та блоки. Він не надає інформації про середовище, як-от операційну систему хоста, IP-адресу чи навіть час. Розробник смарт-контракту намагатиметься знайти інформацію в полі позначки часу в метаданих блоку. На жаль, поле позначки часу блоку є довільним і майнер блоку може записати будь-яку мітку часу, яку він хоче, без жодної перевірки з боку інших вузлів у мережі. Якщо смарт-контракт покладається на таку інформацію про мітку часу, його може обдурити зловмисний майнер.

Залежність від порядку транзакції в блоці (Transaction Ordering Dependency).

Кожна операція в розумному контракті є транзакцією, і навіть якщо кілька транзакцій не виконуються паралельно, порядок цих операцій може дати різні результати. Розглянемо приклад випадку: припустимо, що смарт-

контракт має запис про те, що Аліса спочатку має 0 токенів на своєму рахунку, і вона виконує дві послідовні операції:

– зняття (1000), депозит (1000).

Якщо транзакції впорядковані як

зняти (1000), депозит (1000)

у Аліси не буде достатньо балансу для зняття, і перша операція завершиться невдало, а друга пройде, і в Аліси залишиться 1000 жетонів. Якщо порядок зворотний, обидві транзакції будуть завершені, і на рахунку Аліси залишиться 0 токенів. Майнери відповідають за збір транзакцій і створення блоку. У зловмисного майнера може бути можливість переорганізовувати залежні транзакції таким чином, щоб результат пішов йому на користь.

Використання send. send — це функція Solidity для передачі ефіру зі смарт-контракту на зовнішню адресу. Якщо зовнішня адреса є смарт-контрактом, вона виконуватиме додаткову функцію, виконуючи код резервної функції у зовнішній адресі. Тому, використання функції send в смарт-контракті може призвести до вразливості, і розробник може використовувати замість неї функцію передачі, якщо хоче передати Ether і заборонити виконання коду.

Неперевірені виклики. Виконання коду в зовнішньому смарт-контракті може виконуватися функціями send, call та delegatecall. Однак такі функції не зупиняють виконання та не викликають жодних помилок, якщо викликаний контракт має помилку виконання. Натомість функції повертатимуть false і продовжуватимуть виконувати контракт походження. Таким чином, це може призвести до помилкового потоку виконання. Наприклад, розглянемо наступний код для виходу з символічного контракту:

```
msg.sender.send( amount );
wallet[ msg.sender ] -= amount;
```

Якщо надсилання в першому рядку не вдається, Ether не буде належним чином надіслано на адресу msg.sender, і його буде успішно вираховано з балансу гаманця у другому рядку. Розробник мав поставити «умову if» навколо функції send/call/delegatecall, щоб обробити випадок, коли вона не вдається.

Відмова в обслуговуванні з Throw. Розумний контракт, який покладається на результат зовнішнього контракту, може бути вразливим через цю вразливість. Прикладом є гра King of Ether Throne (King of Ether Throne, 2016). Ця гра вимагає, щоб новий король ставив більшу суму, ніж поточний король. Якщо це вдається, контракт надсилає суму на адресу поточного короля, певну плату розробникам і, нарешті, встановлює нового короля. Однак, якщо поточна адреса короля є смарт-контрактом, і його резервна функція видає команду throw, щоб зупинити потік виконання, що означає відмову отримати будь-який ефір. Плата нинішньому королю ніколи не вдасться, і ніхто не зможе повалити поточного короля.

Обмеження використання GAS та циклічні виклики. Ethereum дуже піклується про обчислення та сховище, необхідні для виконання кожної операції. Отже, користувач, який викликає смарт-контракт, повинен заплатити певну кількість ефіру як «GAS», щоб допомогти фінансувати операції. Чим більше обчислень або пам’яті потрібно для кожної операції, тим більший потрібний газ. Необізнаний розробник може написати функцію смарт-контракту, що містить операції з високою вартістю газу, які не вдасться викликати, якщо абонент не надасть достатньо газу. Розглянемо приклад нижче.

```
address [] accounts ;
uint public fixedInterest = 0.0001 ether ;
function distributeInterest () public {
   require ( msg . sender == owner );
   for ( uint i =0; i < accounts . length ; i ++) {
      accounts [i ]. transfer ( fixedInterest );
   }
}
```

У цьому прикладі розробник хоче розподілити відсотки на кожну адресу, збережену в змінній accounts. Виклик цієї функції може вийти з ладу, коли кількість адрес зростає, а вартість циклічного проходження функції transfer, високовартісної функції, стає надто високою.

2. ПЛАНУВАННЯ АУДИТУ СМАРТ КОНТРАКТУ

Смарт контракти стосуються фінансових активів високої вартості, особливо в таких галузях, як децентралізоване фінансування (DeFi), і цінних предметів, таких як незамінні токени (NFT). Таким чином, незначні вразливості в смарт-контрактах можуть і часто призводять до великих, безповоротних втрат для користувачів. Проте всебічне тестування може виявити помилки в коді смарт-контракту та зменшити ризики безпеки перед розгортанням.

Смарт контракти, розгорнуті у віртуальній машині Ethereum (EVM), незмінні за замовчуванням. У той час як традиційні розробники можуть звикнути виправляти помилки програмного забезпечення після запуску, розробка Ethereum залишає мало місця для виправлення недоліків безпеки після того, як смарт-контракт запрацює на блокчейні.

Хоча механізми оновлення для смарт-контрактів, такі як шаблони проксі, їх може бути важко реалізувати. Окрім зменшення незмінності та введення складності, оновлення часто вимагають складних процесів керування.

Здебільшого оновлення слід вважати останнім заходом і уникати його, якщо це не необхідно. Виявлення потенційних вразливостей і недоліків у вашому смарт-контракті на етапі перед запуском зменшує потребу в оновленні логіки.

2.1 Види тестування і перевірки

Функціональне тестування. Функціональне тестування перевіряє функціональність смарт-контракту та дає впевненість, що кожна функція в коді працює належним чином. Функціональне тестування вимагає розуміння того, як ваш смарт-контракт повинен поводитися в певних умовах. Потім ви

можете перевірити кожну функцію, виконавши обчислення з вибраними значеннями та порівнявши отримані результати з очікуваними результатами.

Функціональне тестування охоплює три методи: модульне тестування, інтеграційне тестування та системне тестування.

Модульне тестування. Модульне тестування передбачає тестування окремих компонентів смарт-контракту на коректність. Модульний тест простий, швидкий у виконанні та дає чітке уявлення про те, що пішло не так, якщо тест провалиться.

Модульні тести мають вирішальне значення для розробки смарт-контрактів, особливо якщо вам потрібно додати нову логіку до коду. Ви можете перевірити поведінку кожної функції та підтвердити, що вона виконується належним чином.

Виконання модульного тесту часто вимагає створення тверджень — простих неофіційних тверджень із визначенням вимог до смарт-контракту. Потім можна використовувати модульне тестування, щоб перевірити кожне твердження та перевірити, чи вірно воно під час виконання.

Приклади контрактних тверджень включають:

«Тільки адмін може призупинити контракт»;

«Не адміністратори не можуть карбувати нові токени»;

«Договір скасовується через помилки»;

Інтеграційне тестування.

Інтеграційне тестування є вищим рівнем, ніж модульне тестування в ієрархії тестування. Під час інтеграційного тестування окремі компоненти смарт-контракту тестуються разом.

Цей підхід виявляє помилки, що виникають через взаємодію між різними компонентами контракту або кількома контрактами. Ви повинні використовувати цей метод, якщо у вас є складний контракт із кількома функціями або той, який взаємодіє з іншими контрактами.

Тестування інтеграції може бути корисним для забезпечення належної роботи таких речей, як успадкування та впровадження залежностей.

Системне тестування. Системне тестування є завершальним етапом функціонального тестування смарт-контрактів. Система оцінює смарт-контракт як один повністю інтегрований продукт, щоб перевірити, чи працює він відповідно до технічних вимог.

Ви можете розглядати цей етап як перевірку наскрізного потоку вашого смарт-контракту з точки зору користувача. Хороший спосіб виконати системне тестування смарт-контракту — це розгорнути його в середовищі, схожому на виробництво, наприклад у тестовій мережі або мережі розробки.

Тут кінцеві користувачі можуть виконувати пробні запуски та повідомляти про будь-які проблеми з бізнес-логікою контракту та загальною функціональністю. Тестування системи є важливим, оскільки ви не можете змінити код після розгортання контракту в основному середовищі EVM.

Статичний/динамічний аналіз. Статичний аналіз і динамічний аналіз є двома автоматизованими методами тестування для оцінки якості безпеки смарт-контрактів. Однак обидва методи використовують різні підходи для пошуку дефектів у коді контракту.

Статичний аналіз. Статичний аналіз перевіряє вихідний код або байт-код смарт-контракту перед виконанням. Це означає, що ви можете налагоджувати код контракту без фактичного запуску програми. Статичні аналізатори можуть виявляти типові вразливості в смарт-контрактах Ethereum і сприяти дотриманню найкращих практик.

Динамічний аналіз. Методи динамічного аналізу вимагають виконання смарт-контракту в середовищі виконання, щоб виявити проблеми у вашому коді. Аналізатори динамічного коду спостерігають за поведінкою контракту під час виконання та створюють детальний звіт про виявлені вразливості та порушення властивостей.

Фаззинг є прикладом техніки динамічного аналізу для тестування контрактів. Під час фаз-тестування фаззер заповнює ваш смарт-контракт неправильними та недійсними даними та відстежує, як контракт реагує на ці вхідні дані.

Як і будь-яка програма, смарт-контракти покладаються на вхідні дані, надані користувачами для виконання функцій. І хоча ми припускаємо, що користувачі вводитимуть правильні дані, це не завжди так.

У деяких випадках надсилання неправильних вхідних значень у смарт-контракт може спричинити витік ресурсів, збої або, що ще гірше, призвести до ненавмисного виконання коду. Fuzzing виявляє такі проблеми заздалегідь, що дозволяє усунути вразливість.

2.2. Інструменти автоматичного аналізу вразливостей

У цьому розділі розглянуті 10 основних інструментів аналізу безпеки, які використовуються для пошуку вразливостей у смарт контрактах. Більшість інструментів в основному використовуються для статичного та динамічного аналізу кодів смарт-контрактів.

Slither. Slither – це фреймворк статичного аналізу коду смарт-контракту. Його методи безпеки виявлення потенційних помилок є швидкими та надійними. Slither можна використовувати для виконання основних завдань, таких як автоматичне виявлення вразливостей, автоматичне виявлення оптимізації, розуміння коду та допоміжний перегляд коду. Для аналізу безпеки запускається багатоетапна процедура. Компілятор Solidity створює абстрактне синтаксичне дерево Solidity (AST) із вихідного коду контракту, і AST використовується як вхідні дані для Slither. На початковому етапі Slither отримує важливу інформацію про контракт, таку як граф успадкування, граф потоку керування (CFG) тощо. Наступний етап включає перетворення повного коду на SlithIR. На наступному етапі завдання аналізу коду виконується шляхом обчислення списку попередньо визначених аналізів.[14]

MythX. MythX – це платний сервіс, який сканує смарт-контракти на основі EVM на наявність вразливостей. Він містить різні методи аналізу, які

включають статичне, динамічне та символічне виконання. Основною метою MythX є підтримка розробників DApp у розробці смарт-контрактів для забезпечення безпечнішої платформи. MythX не відповідає вимогам сам по собі, скоріше він інтегрований із інструментами розробки, такими як Truffle і Remix. Він сумісний не тільки з платформою Ethereum – розробники, пов'язані з Tron, Vechain, Quorum, Roostock і кількома іншими платформами на основі EVM, також можуть скористатися перевагами цих інструментів безпеки, щоб знайти помилки в смарт-контракті. MythX проходить три етапи для аналізу коду смарт-контракту. По-перше, він вимагає від розробників надати свій код; по-друге, потрібно активувати повний набір методів аналізу; нарешті, він створює звіт аналізу, який демонструє наявність будь-яких помилок.[15]

Mythril – це інструмент безпеки, який аналізує смарт-контракти, написані Solidity. Mythril, інструмент із відкритим кодом, використовує переваги техніки символічного виконання для визначення помилок у коді. Перевірка недоліків безпеки передбачає виконання байт-коду смарт-контракту в спеціально створеному EVM. Щоб виконати аналіз безпеки, Mythril проходить чотири основні робочі етапи. Коли виявляється недолік у програмі, вхідні транзакції аналізуються для визначення можливих причин. Цей метод безпеки допомагає визначити основну причину вразливості програми, а також пом'якшити експлуатацію. Якщо розробник створює вихідний код контракту, Mythril може знайти помилки в коді. [16]

Manticore – це інструмент аудиту Solidity, який виконує символічний аналіз смарт-контрактів. Основні функції manticore включають трасування вхідних даних, які завершують програму, реєстрацію реалізації інструкцій на рівні та надання доступу до механізму аналізу через Python API. Він має функцію динамічного символічного виконання, яка аналізує двійкові файли, а також смарт-контракти Ethereum. Основні атрибути в архітектурі Manticore

включають Core Engine, Native Execution Modules і Ethereum Execution Modules. Модуль Satisfiability Modulo Theories (SMT-LIB), система подій і API розглядаються як вторинні атрибути. [17]

Securify – це інструмент аналізу безпеки смарт-контрактів. Securify — це автоматизований інструмент, який на основі наданих атрибутів може визначити, чи виконується контракт належним чином. Securify — це продукт з відкритим кодом, функція аналізу безпеки якого проходить два етапи для виконання необхідного завдання. Securify приймає байт-код EVM для аналізу безпеки. Контракти, написані в Solidity, також приймаються як вхідні дані, однак код має бути скомпільований у байт-код EVM, щоб задіяти процес безпеки. Коли ініціюється порушення безпеки, Securify створює команду, яка спонукає шаблон порушення до відповідності. Подібним чином, якщо і шаблон порушення, і шаблон відповідності не збігаються, створюється попередження. Техніка аналізу безпеки Securify є унікальною в порівнянні з іншими інструментами. Securify використовує статичний аналіз для аналізу кожного шляху смарт-контракту. [18][19]

SmartCheck. SmartCheck – це автоматизований розширений інструмент аналізу вразливостей для смарт-контрактів Solidity. SmartCheck — це система з відкритим вихідним кодом, яка не лише вказує на вразливі місця в коді смарт-контракту, але й уточнює причини вразливостей за допомогою відповідного опису та рекомендацій. SmartCheck було реалізовано шляхом використання запитів XPath [xpa] у проміжному представленні (IR) для виявлення моделей уразливостей. SmartCheck захищає будь-який проаналізований код, який було перетворено на IR, а пов’язані з ним елементи визначаються за допомогою XPath-відповідності. SmartCheck започаткував експеримент з безпеки на понад 4600 дійсних контрактах. Було встановлено, що 86,6% контактів складали нульовий баланс, тоді як один договір складав баланс лише на 38,4% від загального балансу. Аналіз SmartCheck показав, що 99,9% проаналізованих контрактів містили певний

дефект безпеки, при цьому 63,2% контрактів були серйозно вразливими. [20]

Echidna. Echidna — смарт-фаззер EVM, який визначає помилки в коді Solidity. Цей інструмент потребує лише пропозицій Solidity для проведення глибокого аналізу помилок і забезпечує зрозумілий інтерфейс користувача (UI) для спрощення результатів. Єхидна використовує різні комбінації вхідних даних, поки їй не вдасться порушити надану властивість. Єхидна містить кілька подібних атрибутів до Manticore, що дозволяє їй функціонувати на рівні EVM. Крім того, його також можна консолідувати для безперервної інтеграції (CI), щоб виявити помилки коду під час розробки. Echidna надає безліч інструментів для створення індивідуальних аналізів для роботи зі складними контрактами. Цей інструмент використовує стек, тому необхідна залежність базуватиметься на версії solc, яку використовує контракт. [21]

Oyente. Oyente – символічний інструмент виконання, який використовується для пошуку помилок безпеки в смарт-контрактах. Oyente вивчає смарт-контракти Ethereum, щоб виявити лазівки в безпеці, які можуть спричинити потенційні загрози. Oyente не тільки виявляє небезпечні помилки, але й досліджує кожен практичний шлях виконання. Експеримент, проведений Oyente на 19 366 смарт-контрактах, призвів до того, що 8 833 з них були визначені як вразливі. Символічний метод виконання символічно представляє природу шляху виконання у вигляді математичної формули. OYENTE проводить порівняння між новою формулою та формулами, які містять звичайні помилки, щоб з’ясувати, чи обидві формули дійсні одночасно. [22]

Vandal. Vandal – ще одна структура аналізу безпеки для смарт-контрактів. Vandal містить конвеєр аналізу, який перетворює байт-код EVM у семантичні логічні відносини. Vandal — це дуже швидкий і ефективний інструмент аналізу безпеки, який перевірив понад 95% із 141000 смарт-контрактів із середнім часом виконання лише 4,15 секунди. Низькі накладні витрати перевищують загальну продуктивність основних існуючих

інструментів аналізу безпеки. Дизайн безпеки Vandal містить декларативну мову під назвою Souffé. Виконання аналізу безпеки декларативною мовою допомагає аналітикам безпеки отримати прототип останнього аналізу. [23]

Zeus. Zeus – це практична основа для перевірки дійсності смарт-контрактів. Він використовує переваги абстрактної інтерпретації та перевірки символічної моделі для аналізу безпеки смарт-контрактів. Прототип Zeus протестував понад 22400 смарт-контрактів, показавши, що близько 94,6% цих контрактів є вразливими. Zeus приймає код смарт-контракту та генерує автентичну версію в шаблоні в стилі XACML. Код смарт-контракту та специфікації політики транслюються в біт-код LLVM для покращення поведінки контракту. Zeus виконує статичний аналіз наданого коду смарт-контракту, щоб додати політику заяви assert у потрібному місці програми. [24]

2.3 Ручний аналіз вразливостей

Ручне тестування здійснюється за допомогою людини та передбачає участь особи, яка виконує кроки тестування вручну. Аудити коду, коли розробники та/або аудитори перевіряють кожен рядок коду контракту, є прикладом ручного тестування смарт-контрактів. Ручне тестування смарт-контрактів вимагає значних навичок і значних витрат часу, грошей і зусиль. Крім того, ручне тестування іноді може бути вразливим до проблем людської помилки.

Однак застосування ручного тестування до смарт-контрактів також може бути корисним. Аудит коду використовує людський інтелект для пошуку дефектів у коді контракту, які можуть залишитися непоміченими під час автоматизованого тестування.

Ручне тестування смарт-контрактів також може виявити вразливості, які існують поза кодом, але все одно можуть впливати на нього. Наприклад,

аудит смарт-контракту може виявити вразливі місця, що виникають через неправильну взаємодію з компонентами поза мережею.

Аудити коду. Аудит коду — це детальна оцінка вихідного коду смарт-контракту для виявлення можливих точок збою, недоліків безпеки та поганих практик розробки. Хоча перевірки коду можна автоматизувати, тут ми маємо на увазі аналіз коду за допомогою людини.

Аудит коду потребує мислення зловмисника, щоб визначити можливі вектори атак у смарт-контрактах. Навіть якщо запускається автоматизований аудит, аналіз кожного рядка вихідного коду є мінімальною вимогою для написання безпечних смарт-контрактів.

Формальна перевірка. Хоча тестування допомагає підтвердити, що контракт повертає очікувані результати для деяких вхідних даних, воно не може остаточно підтвердити те саме для вхідних даних, які не використовувалися під час тестування. Тестування смарт-контракту не може гарантувати «функціональну коректність», тобто не може показати, що програма поводиться відповідно до всіх наборів вхідних значень і умов.

Таким чином, розробникам рекомендується включити формальну перевірку в свій підхід до оцінки правильності смарт-контрактів. Формальна перевірка використовує формальні методи — математично точні методи визначення та перевірки програмного забезпечення.

Формальна перевірка вважається важливою для смарт-контрактів, оскільки вона допомагає розробникам офіційно перевірити припущення, що стосуються смарт-контрактів. Це робиться шляхом створення формальних специфікацій, які описують властивості смарт-контракту, і перевірки того, що формальна модель смарт-контракту відповідає специфікації. Такий підхід підвищує впевненість у тому, що смарт-контракт виконуватиме лише функції, визначені його бізнес-логікою, і нічого більше.

3 РЕАЛІЗАЦІЯ АУДИТУ СМАРТ-КОНТРАКТУ

Для аудиту безпеки буде використано смарт контракт токену TEX на базі BEP20 (сумісного з ERC-20) для блокчейну BSC. У вимогах до аудиту - провести статичний аналіз вихідного коду контракту та функціональну перевірку працездатності з завантаженням в блокчейн.

3.1. Статичний аналіз за допомогою утиліти Slither

Для статичного аналізу буде використано безкоштовну утиліту slither, а саме її реалізацію на базі docker-контейнеру останньої версії:

```
sudo docker pull trailofbits/eth-security-toolbox
sudo docker run -it -v /home/svyatoslav/Projects:/share
trailofbits/eth-security-toolbox
```

Для візуалізації структури контракту можна виконати команду:

```
ethsec@23b4894c76a7:/share/TEX/contracts$ slither TEX.sol --print
inheritance-graph
Inheritance Graph: TEX.sol.inheritance-graph.dot

TEX.sol analyzed (7 contracts)

ethsec@23b4894c76a7:/share/TEX/contracts$ dot TEX.sol.inheritance-
graph.dot -Tsvg -o TEX.svg
```

Зі структури видно, що основний контракт TEX наслідує базовий інтерфейс IBEP20 та Ownable, який в свою чергу наслідує Context.

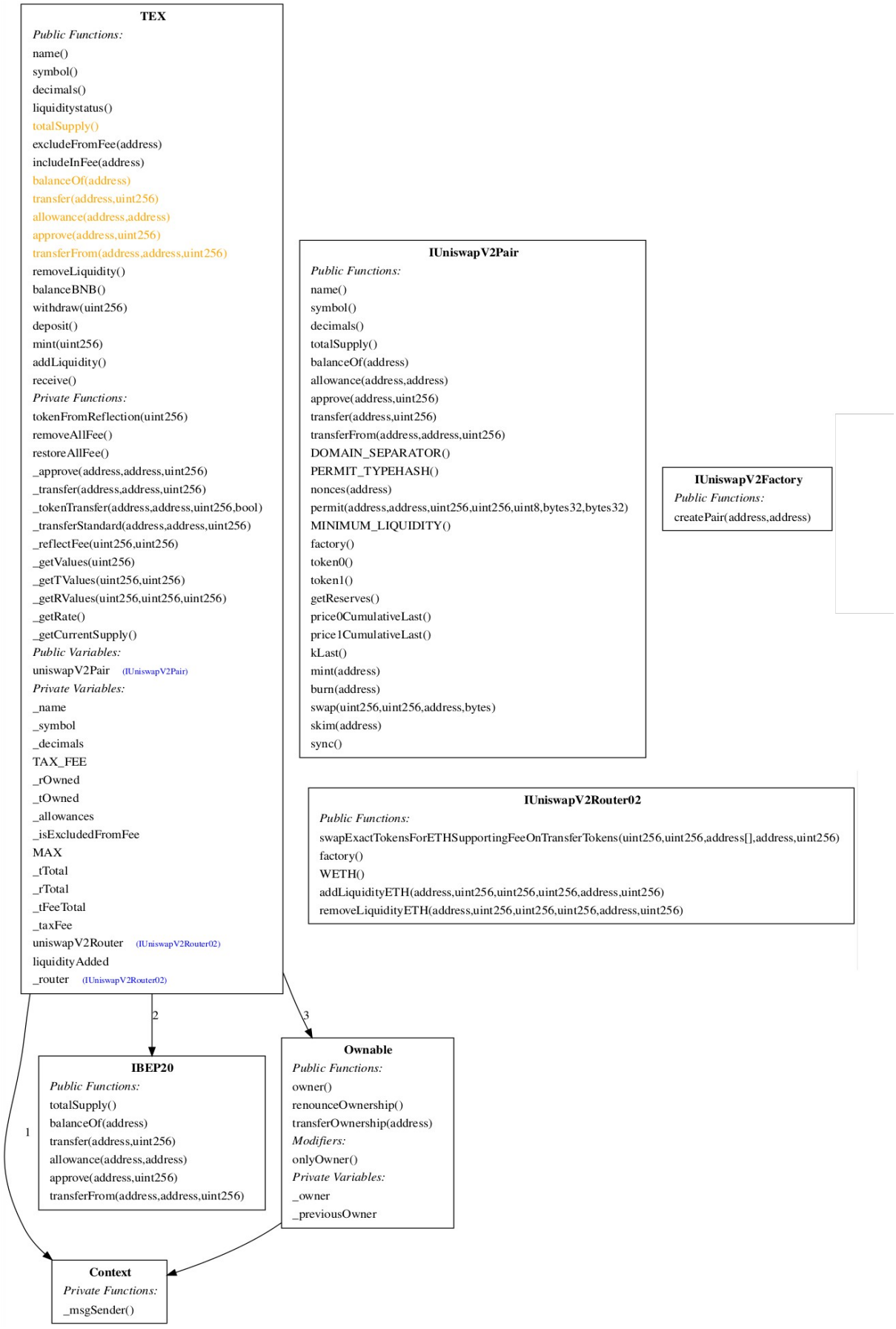


Рисунок 3.1 – Графічна діаграма структури смарт контракту.

Запуск аналізу смарт-контракту (повний звіт в додатку 8.1):

```
ethsec@825857d9688f:/share/TEX$ slither contracts/TEX.sol
TEX.removeLiquidity() (contracts/TEX.sol#253-256) ignores return value
by
_router.removeLiquidityETH(address(this),0,0,0,owner(),block.timestamp
) (contracts/TEX.sol#254)
TEX.addLiquidity() (contracts/TEX.sol#272-284) ignores return value by
_router.addLiquidityETH{value: msg.value}
(address(this),amount,0,0,msg.sender,block.timestamp)
(contracts/TEX.sol#276-281)
TEX.addLiquidity() (contracts/TEX.sol#272-284) ignores return value by
uniswapV2Pair.approve(address(_router),type()(uint256).max)
(contracts/TEX.sol#283)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#unused-return

...
balanceBNB() should be declared external:
    - TEX.balanceBNB() (contracts/TEX.sol#257-259)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#public-function-that-could-be-declared-external
contracts/TEX.sol analyzed (7 contracts with 78 detectors), 44
result(s) found
```

Результатом є реакція 44 детекторів, з яких:

Рівень	Кількість детекторів	
Medium	3	
Low	5	
Informational	19	
Optimization	17	

Перевірка на відповідність стандарту ERC-20:

```
ethsec@c20081cbfa90:~$ slither-check-erc /share/TEX/contracts/TEX.sol
TEX
# Check TEX

## Check functions
[✓] totalSupply() is present
   [✓] totalSupply() -> (uint256) (correct return type)
   [✓] totalSupply() is view
[✓] balanceOf(address) is present
   [✓] balanceOf(address) -> (uint256) (correct return type)
   [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
```

```
   [✓] transfer(address,uint256) -> (bool) (correct return type)
   [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
   [✓] transferFrom(address,address,uint256) -> (bool) (correct return type)
   [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
   [✓] approve(address,uint256) -> (bool) (correct return type)
   [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
   [✓] allowance(address,address) -> (uint256) (correct return type)
   [✓] allowance(address,address) is view
[✓] name() is present
   [✓] name() -> (string) (correct return type)
   [✓] name() is view
[✓] symbol() is present
   [✓] symbol() -> (string) (correct return type)
   [✓] symbol() is view
[✓] decimals() is present
   [✓] decimals() -> (uint8) (correct return type)
   [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
   [✓] parameter 0 is indexed
   [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
   [✓] parameter 0 is indexed
   [✓] parameter 1 is indexed


    [ ] TEX is not protected for the ERC20 approval race condition
```

Згідно статистичного аналізу контракт не має критичних зауважень, в більшості відповідає стандарту ERC-20, але потребує внесення змін для усунення недоліків і економії комісії за використання GAS і усунення проблеми залежності від порядку розміщення транзакцій в блоці.

3.2 Функціональне тестування

Для функціонального тестування необхідно забезпечити роботу контракту разом з іншими смарт-контрактами, які вже розгорнуті в блокчейні. Оскільки смарт-контракт орієнтований на роботу з біржею

Pancakeswap, то всі необхідні контракти треба взяти з вказаних адрес реального блокчейну https://docs.pancakeswap.finance/code/smart-contracts

PancakeFactory
https://bscscan.com/address/0xca143ce32fe78f1f7019d7d551a6402fc5350c73#code

PancakeRouter
https://bscscan.com/address/0x10ed43c718714eb63d5aa57b78b54704e256024e#code

Крім того, для функціонального тестування обміну на біржі необхідна відповідна пара токена **WBNB**:
https://bscscan.com/token/0xbb4cdb9cbd36b01bd1cbaebf2de08d9173bc095c

Всі ці контракти скомпільовані під різними версіями solidity, тому замість компілювання вихідного коду і для пришвидшення завантаження контрактів в тестовий блокчейн буде використовувати готовий байткод, який необхідно вставити у відповідні константи відповідних модулів тесту - PancakeRouter.js, PancakeFactory.js, WBNB.js.

Для **PancakePair** достатньо скопілювати контракт, щоб отримати його ABI.

Функціональний тест включає в себе такі функції:

1. Запуск локальної ноди Ethereum на базі ganache-cli
2. Завантаження байткоду WBNB в ноду
3. Завантаження байткоду PancakeFactory в ноду
4. Завантаження байткоду PancakeRouter в ноду
5. Компіляція контракту TEX.sol
6. Завантаження байткоду TEX в ноду
7. Створення ліквідності 10млн TEX : 10BNB на обміннику PancakeRouter
8. Отримання адреси створеного контракту PancakePair
9. Купівля через обмінник токенів TEX на 1 BNB

10.Випуск токенів TEX контрактом TEX

11.Вилучення ліквідності на обміннику PancakeRouter

Після підготовки всіх контрактів і запуску тесту на тестовій ноді блокчейну отримано результат виконання тесту:

```
 > Testbox@1.0.0 test
 > echo Wait 10 sec before ETH node starts && sleep 10 && mocha --
timeout 300000

 Wait 10 sec before ETH node starts


   TEST BOX
      ✓ Deploy WBNB (382ms)
      ✓ Deploy PancakeFactory (437ms)
      ✓ Deploy PancakeRouter (418ms)
      ✓ Compile TEX (10712ms)
      ✓ Deploy TEX (321ms)
      ✓ Add Liquidity (819ms)
      ✓ Get uniswapV2Pair (58ms)

 Address    BNB    WBNB    TEX    LP
 contract     0    0    0    10000
 Pair       0    10    10000000    0
 WBNB       10    0    0    0
 owner      89.704    0    0    0
 user1      100    0    0    0
 user2      100    0    0    0
 user3      100    0    0    0
 user4      100    0    0    0
 Token price: 0.000000997499900499 BNB
      ✓ Start balances (1559ms)

 Address    BNB    WBNB    TEX    LP
 contract     0    0    0    10000
 Pair       0    11    9092975.677    0
 WBNB       11    0    0    0
 owner      89.704    0    0    0
 user1      98.997    0    907024.324    0
 user2      100    0    0    0
 user3      100    0    0    0
 user4      100    0    0    0
 Token price: 0.000001206700555124 BNB
      ✓ Buy Token for BNB (1682ms)


 Address    BNB    WBNB    TEX    LP
```

```
contract     0.963    0    0    10016.324
Pair       0    11.018    9107818.175    0
WBNB       11.018    0    0    0
owner      89.724    0    0    0
user1      98.997    0    907024.324    0
user2      98.996    0    727282.425    0
user3      100    0    0    0
user4      100    0    0    0
Token price: 0.000001206700555534 BNB
      ✓ Mint tokens for user2 (1527ms)

Address    BNB    WBNB    TEX    LP
contract     0    0    9107818.175    0
Pair       0    0.001    0.001    0
WBNB       0.001    0    0    0
owner      101.701    0    0    0
user1      98.997    0    907024.324    0
user2      98.996    0    727282.425    0
user3      100    0    0    0
user4      100    0    0    0
Token price: 0.000000000000000001 BNB
      ✓ Admin close project, remove liquidity (1613ms)


  11 passing (20s)
```

Функціональний тест підтверджує працездатність токену TEX при роботі з іншими контрактами на базі обмінника Pancakeswap. Враховуючи наявність можливості вилучення ліквідності, даний контракт може бути використаний в якості зловмисного інструменту з метою акумуляції коштів та раптового припинення забезпечення ліквідності в базовій криптовалюті BNB. Таким чином всі власники токену TEX не будуть мати можливості обміняти свої TEX токени назад на BNB.

3.3. Системні підходи в забезпеченні безпеки блокчейн проектів

Ось перелік важливих причин вибору аудиту смарт-контрактів як обов’язкового компонента стратегій безпеки смарт-контрактів.

- Ранні перевірки розумного коду можуть допомогти уникнути небажаних витрат через помилки.

- аудитори безпеки можуть надати експертну інформацію та рецензії на код.
- Часті оцінки безпеки можуть сприяти покращенню середовища розробки.
- Аудити смарт-контрактів допомагають проактивно ідентифікувати ризики безпеки для кодів смарт-контрактів.
- Часті аудити для смарт-контрактів протягом життєвого циклу розробки можуть допомогти отримати аналітичну інформацію, таку як резюме або деталі вразливостей.

Проведення аудиту безпеки смарт-контрактів. Обговорення найкращих практик безпеки смарт-контрактів також висвітлить етапи простого аудиту смарт-контрактів. Навіть якщо різні аудитори можуть включити унікальні моменти у свої підходи, наступні кроки є частиною стандартної процедури.

Колекція моделей для розробки коду. Аудитори збирають деталі щодо специфікації коду, а потім перевіряють архітектуру для забезпечення інтеграції смарт-контрактів сторонніх розробників. Цей крок має вирішальне значення для того, щоб допомогти аудиторам зрозуміти різні цілі та масштаби проекту.

Використання модульних тестів. Процес аудиту безпеки смарт-контракту вимагає виконання модульних тестів. Аудитори перевірятимуть кожну функцію смарт-контракту, щоб визначити її зручність використання. На цьому кроці аудитори покладаються на ручні та автоматизовані інструменти для включення загального коду смарт-контракту в випадки модульного тестування.

Вибір методу аудиту. Аудит смарт-контракту наголошує на виборі методів аудиту, оскільки ручний і автоматизований аудит мають явні переваги. У більшості випадків ручні аудити можуть бути ефективнішими порівняно з автоматизованими. Аудитори не повинні покладатися на будь-яке програмне забезпечення з ручним аудитом смарт-контрактів, а також

можуть виявляти атаки, такі як залежності від порядку розміщення транзакцій в блоці.

Складання та публікація аудиторського звіту. Останнім етапом аудиту смарт-контракту є складання початкового звіту. Після завершення першого етапу аудиту аудитори окреслять проблеми з кодом і нададуть рекомендації щодо усунення помилок. Після виправлення помилок аудитори мають підготувати остаточний звіт, у якому відображено заходи щодо виправлення помилок, які впровадила команда проекту.

ВИСНОВКИ

Технології смарт-контрактів дозволяють користувачам формувати децентралізовані цифрові угоди без участі третьої сторони. Технологія розумних контрактів залучила такі сектори, як охорона здоров’я, управління бізнесом, акціонерні угоди та страхування. Однак чим більше ця технологія поширюється, тим більше вона привертає увагу потенційних зловмисників, що призводить до кількох серйозних дій.

Потреба в безпеці у випадку смарт-контрактів стає все більш глибокою з кожним днем. З огляду на багато проблем щодо безпеки, неефективності та невідповідної поведінки, безпека смарт-контракту залишається під сумнівом. Тривіальні помилки в коді смарт-контракту можуть коштувати організації мільйони або навіть мільярди доларів. Тому аудит безпеки смарт-контрактів став однією з обов’язкових вимог перед розгортанням смарт-контрактів.

Дослідження вказує на те, що належне рішення для захисту смарт-контрактів залишається проблемою, і майбутня робота включатиме розробку стратегій для виявлення та пом’якшення основних недоліків безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Список криптовалют. [Електронний ресурс].- Режим доступу: https://coinmarketcap.com/currencies/ethereum/

2. Nick Szabo -- Smart Contracts: Building Blocks for Digital Markets. [Електронний ресурс].- Режим доступу: https://web.archive.org/web/20180427165653/http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

3. Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. [Електронний ресурс].- Режим доступу: https://books.google.com.ua/books?id=roVkDgAAQBAJ&pg=PA72&lpg=PA72&dq=Solidity+LLL+Mutan&source=bl&ots=G5NhxFf0zR&sig=ACfU3U02dsde7Zh3ynzflwwebT8pBPmLeg&hl=ru&sa=X&ved=2ahUKEwii3vDytp_pAhVdAhAIHbI7BJIQ6AEwBHoECAoQAQ#v=onepage&q=Solidity LLL Mutan&f=false.

4. Smart Contracts, Explained. [Електронний ресурс].- Режим доступу: https://web.archive.org/web/20180121073949/https://cointelegraph.com/explained/smart-contracts-explained

5. Exploring Vulnerabilities in Solidity Smart Contract, Phitchayaphong Tantikul and Sudsanguan Ngamsuriyaroj. [Електронний ресурс].- Режим доступу: https://www.scitepress.org/Papers/2020/89098/89098.pdf

6. Smart Contract Vulnerability Detection Model Based on Multi-Task Learning, Jing Huang, Kuo Zhou, Ao Xiong and Dongmeng Li. [Електронний ресурс].- Режим доступу: https://mdpi-res.com/d_attachment/sensors/sensors-22-01829/article_deploy/sensors-22-01829-v2.pdf?version=1645873574

7. Ethereum Smart Contracts: Security Vulnerabilities and Security Tools, Ardit Dika [Електронний ресурс].- Режим доступу:

https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2479191/18400_FULLTEXT.pdf

8. Smart Contract Attacks and Protections. [Електронний ресурс].- Режим доступу: https://www.researchgate.net/publication/338926064_Smart_Contract_Attacks_and_Protections

9. Перелік токенів Ethereum. [Електронний ресурс].- Режим доступу: https://etherscan.io/tokens

10. Cryptokitties [Електронний ресурс].- Режим доступу: https://www.cryptokitties.co/

11. McDonald's has filed a trademark for a restaurant in the metaverse that will actually deliver food to your home. [Електронний ресурс].- Режим доступу: https://www.businessinsider.com/mcdonalds-metaverse-virtual-online-restaurant-trademark-delivers-food-web3-nft-2022-2?r=US&IR=T

12. A “Meta” World Built on Blockchain. [Електронний ресурс].- Режим доступу: https://www.finyear.com/A-Meta-World-Built-on-Blockchain_a45800.html

13. Metaverse and money. [Електронний ресурс].- Режим доступу: https://www.citivelocity.com/citigps/metaverse-and-money/?linkId=159079389

14. J. Feist, G. Grieco, and A. Groce, “Slither: a static analysis framework for smart contracts,” in 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019, pp. 8–15

15. MythX, Smart Contract Security Tool for Ethereum, 2019 [Електронний ресурс].- Режим доступу: https://mythx.io/ .

16. B. Mueller, Practical Smart Contract Security Analysis and Exploitation [Електронний ресурс].- Режим доступу: https://medium.com/hackernoon/practical-smart-contract-security-analysis-and-exploitation-part-1-6c2f2320b0c .

17. H. Official, Introduction to Manticore, a symbolic analysis tool for smart contract [Електронний ресурс].- Режим доступу: https://medium.com/haloblock/introduction-to-manticore-a-symbolic-analysis-tool-for-smart-contract-9de08dae4e1e

18. Securify, Securify: Security Scanner for Ethereum Smart Contracts, [Електронний ресурс].- Режим доступу: https://securify.chainsecurity.com/ .

19. P. Tsankov, A. M. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. T. Vechev, “Securify: Practical security analysis of smart contracts,” [Електронний ресурс].- Режим доступу: http://arxiv.org/abs/1806.01143

20. Smartdec, SmartCheck [Електронний ресурс].- Режим доступу: https://tool.smartdec.net/ .

21. trailofbits.com, Echidna, a smart fuzzer for Ethereum, [Електронний ресурс].- Режим доступу: https://blog.trailofbits.com/2018/03/09/echidna-a-smart-fuzzer-for-ethereum/ .

22. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS ’16. New York, NY, USA: ACM, 2016, pp. 254–269. [Електронний ресурс].- Режим доступу: http://doi.acm.org/10.1145/2976749.2978309

23. L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz, “Vandal: A scalable security analysis framework for smart contracts,” [Електронний ресурс].- Режим доступу: http://arxiv.org/abs/1809.03981

24. S. Jarzabek, A. Poniszewska-Mara´nda, and L. Madeyski, Integrating Research and Practice in Software Engineering, ser. Studies in computational intelligence. Springer International Publishing, 2019. [Електронний ресурс].- Режим доступу: https://books.google.co.uk/books?id=LR2nDwAAQBAJ

ДОДАТОК А

Результат виконання сканування утилітою Slither

```
ethsec@825857d9688f:/share/TEX$ slither contracts/TEX.sol
TEX.removeLiquidity() (contracts/TEX.sol#253-256) ignores return value
by
_router.removeLiquidityETH(address(this),0,0,0,owner(),block.timestamp
) (contracts/TEX.sol#254)
TEX.addLiquidity() (contracts/TEX.sol#272-284) ignores return value by
_router.addLiquidityETH{value: msg.value}
(address(this),amount,0,0,msg.sender,block.timestamp)
(contracts/TEX.sol#276-281)
TEX.addLiquidity() (contracts/TEX.sol#272-284) ignores return value by
uniswapV2Pair.approve(address(_router),type()(uint256).max)
(contracts/TEX.sol#283)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#unused-return

TEX.allowance(address,address).owner (contracts/TEX.sol#203) shadows:
    - Ownable.owner() (contracts/TEX.sol#81-83) (function)
TEX._approve(address,address,uint256).owner (contracts/TEX.sol#233)
shadows:
    - Ownable.owner() (contracts/TEX.sol#81-83) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#local-variable-shadowing

Ownable.constructor().msgSender (contracts/TEX.sol#76) lacks a zero-
check on :
      - _owner = msgSender (contracts/TEX.sol#77)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#missing-zero-address-validation

Reentrancy in TEX.addLiquidity() (contracts/TEX.sol#272-284):
    External calls:
    - uniswapV2Pair =
IUniswapV2Pair(IUniswapV2Factory(_router.factory()).createPair(address
(this),_router.WETH())) (contracts/TEX.sol#275)
    - _router.addLiquidityETH{value: msg.value}
(address(this),amount,0,0,msg.sender,block.timestamp)
(contracts/TEX.sol#276-281)
    External calls sending eth:
    - _router.addLiquidityETH{value: msg.value}
(address(this),amount,0,0,msg.sender,block.timestamp)
(contracts/TEX.sol#276-281)
    State variables written after the call(s):
    - liquidityAdded = true (contracts/TEX.sol#282)
Reentrancy in TEX.removeLiquidity() (contracts/TEX.sol#253-256):
    External calls:
```

```
    -
_router.removeLiquidityETH(address(this),0,0,0,owner(),block.timestamp
) (contracts/TEX.sol#254)
    State variables written after the call(s):
    - liquidityAdded = false (contracts/TEX.sol#255)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#reentrancy-vulnerabilities-2

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (contracts/TEX.sol#36) is
not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (contracts/TEX.sol#37) is
not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (contracts/TEX.sol#54) is
not in mixedCase
Function IUniswapV2Router02.WETH() (contracts/TEX.sol#115) is not in
mixedCase
Constant TEX._name (contracts/TEX.sol#136) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant TEX._symbol (contracts/TEX.sol#137) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant TEX._decimals (contracts/TEX.sol#138) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant TEX._tTotal (contracts/TEX.sol#146) is not in
UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions

Redundant expression "restoreAllFee (contracts/TEX.sol#250)" inTEX
(contracts/TEX.sol#134-344)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#redundant-statements

Variable
TEX._transferStandard(address,address,uint256).rTransferAmount
(contracts/TEX.sol#293) is too similar to
TEX._transferStandard(address,address,uint256).tTransferAmount
(contracts/TEX.sol#293)
Variable
TEX._transferStandard(address,address,uint256).rTransferAmount
(contracts/TEX.sol#293) is too similar to
TEX._getValues(uint256).tTransferAmount (contracts/TEX.sol#310)
Variable TEX._getRValues(uint256,uint256,uint256).rTransferAmount
(contracts/TEX.sol#328) is too similar to
TEX._getTValues(uint256,uint256).tTransferAmount
(contracts/TEX.sol#320)
Variable TEX._getValues(uint256).rTransferAmount
(contracts/TEX.sol#312) is too similar to
TEX._getTValues(uint256,uint256).tTransferAmount
(contracts/TEX.sol#320)
```

```
Variable
TEX._transferStandard(address,address,uint256).rTransferAmount
(contracts/TEX.sol#293) is too similar to
TEX._getTValues(uint256,uint256).tTransferAmount
(contracts/TEX.sol#320)
Variable TEX._getValues(uint256).rTransferAmount
(contracts/TEX.sol#312) is too similar to
TEX._getValues(uint256).tTransferAmount (contracts/TEX.sol#310)
Variable TEX._getRValues(uint256,uint256,uint256).rTransferAmount
(contracts/TEX.sol#328) is too similar to
TEX._getValues(uint256).tTransferAmount (contracts/TEX.sol#310)
Variable TEX._getRValues(uint256,uint256,uint256).rTransferAmount
(contracts/TEX.sol#328) is too similar to
TEX._transferStandard(address,address,uint256).tTransferAmount
(contracts/TEX.sol#293)
Variable TEX._getValues(uint256).rTransferAmount
(contracts/TEX.sol#312) is too similar to
TEX._transferStandard(address,address,uint256).tTransferAmount
(contracts/TEX.sol#293)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#variable-names-are-too-similar

Ownable._previousOwner (contracts/TEX.sol#72) is never used in TEX
(contracts/TEX.sol#134-344)
TEX._tOwned (contracts/TEX.sol#142) is never used in TEX
(contracts/TEX.sol#134-344)
TEX.uniswapV2Router (contracts/TEX.sol#152) is never used in TEX
(contracts/TEX.sol#134-344)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#unused-state-variable

Ownable._previousOwner (contracts/TEX.sol#72) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#state-variables-that-could-be-declared-constant

renounceOwnership() should be declared external:
    - Ownable.renounceOwnership() (contracts/TEX.sol#90-93)
transferOwnership(address) should be declared external:
    - Ownable.transferOwnership(address) (contracts/TEX.sol#95-99)
name() should be declared external:
    - TEX.name() (contracts/TEX.sol#166-168)
symbol() should be declared external:
    - TEX.symbol() (contracts/TEX.sol#170-172)
decimals() should be declared external:
    - TEX.decimals() (contracts/TEX.sol#174-176)
liquiditystatus() should be declared external:
    - TEX.liquiditystatus() (contracts/TEX.sol#178-180)
totalSupply() should be declared external:
    - TEX.totalSupply() (contracts/TEX.sol#182-184)
excludeFromFee(address) should be declared external:
```

```
    - TEX.excludeFromFee(address) (contracts/TEX.sol#186-188)
includeInFee(address) should be declared external:
    - TEX.includeInFee(address) (contracts/TEX.sol#190-192)
transfer(address,uint256) should be declared external:
    - TEX.transfer(address,uint256) (contracts/TEX.sol#198-201)
allowance(address,address) should be declared external:
    - TEX.allowance(address,address) (contracts/TEX.sol#203-205)
approve(address,uint256) should be declared external:
    - TEX.approve(address,uint256) (contracts/TEX.sol#207-210)
transferFrom(address,address,uint256) should be declared external:
    - TEX.transferFrom(address,address,uint256)
(contracts/TEX.sol#212-216)
balanceBNB() should be declared external:
    - TEX.balanceBNB() (contracts/TEX.sol#257-259)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#public-function-that-could-be-declared-external
contracts/TEX.sol analyzed (7 contracts with 78 detectors), 44
result(s) found
```

ДОДАТОК Б

Вихідний код смарт-контракту токену contracts/TEX.sol

```
//SPDX-License-Identifier: MIT

pragma solidity ^0.8.4;

abstract contract Context {
      function _msgSender() internal view virtual returns (address) {
      return msg.sender;
      }
}

interface IBEP20 {
      function totalSupply() external view returns (uint256);
      function balanceOf(address account) external view returns (uint256);
      function transfer(address recipient, uint256 amount) external returns
(bool);
      function allowance(address owner, address spender) external view
returns (uint256);
      function approve(address spender, uint256 amount) external returns
(bool);
      function transferFrom(address sender, address recipient, uint256
amount) external returns (bool);
      event Transfer(address indexed from, address indexed to, uint256
value);
      event Approval(address indexed owner, address indexed spender, uint256
value);
}
interface IUniswapV2Pair {
  event Approval(address indexed owner, address indexed spender, uint value);
  event Transfer(address indexed from, address indexed to, uint value);

  function name() external pure returns (string memory);
  function symbol() external pure returns (string memory);
  function decimals() external pure returns (uint8);
  function totalSupply() external view returns (uint);
  function balanceOf(address owner) external view returns (uint);
  function allowance(address owner, address spender) external view returns
(uint);

  function approve(address spender, uint value) external returns (bool);
  function transfer(address to, uint value) external returns (bool);
  function transferFrom(address from, address to, uint value) external
returns (bool);

  function DOMAIN_SEPARATOR() external view returns (bytes32);
  function PERMIT_TYPEHASH() external pure returns (bytes32);
  function nonces(address owner) external view returns (uint);

  function permit(address owner, address spender, uint value, uint deadline,
uint8 v, bytes32 r, bytes32 s) external;
```

```
  event Mint(address indexed sender, uint amount0, uint amount1);
  event Burn(address indexed sender, uint amount0, uint amount1, address
indexed to);
  event Swap(
      address indexed sender,
      uint amount0In,
      uint amount1In,
      uint amount0Out,
      uint amount1Out,
      address indexed to
  );
  event Sync(uint112 reserve0, uint112 reserve1);

  function MINIMUM_LIQUIDITY() external pure returns (uint);
  function factory() external view returns (address);
  function token0() external view returns (address);
  function token1() external view returns (address);
  function getReserves() external view returns (uint112 reserve0, uint112
reserve1, uint32 blockTimestampLast);
  function price0CumulativeLast() external view returns (uint);
  function price1CumulativeLast() external view returns (uint);
  function kLast() external view returns (uint);

  function mint(address to) external returns (uint liquidity);
  function burn(address to) external returns (uint amount0, uint amount1);
  function swap(uint amount0Out, uint amount1Out, address to, bytes calldata
data) external;
  function skim(address to) external;
  function sync() external;
}

contract Ownable is Context {
      address private _owner;
      address private _previousOwner;
      event OwnershipTransferred(address indexed previousOwner, address
indexed newOwner);

      constructor() {
      address msgSender = _msgSender();
      _owner = msgSender;
      emit OwnershipTransferred(address(0), msgSender);
      }

      function owner() public view returns (address) {
      return _owner;
      }

      modifier onlyOwner() {
      require(_owner == _msgSender(), "Ownable: caller is not the owner");
      _;
      }

      function renounceOwnership() public virtual onlyOwner {
      emit OwnershipTransferred(_owner, address(0));
      _owner = address(0);
```

```
      }

      function transferOwnership(address newOwner) public virtual onlyOwner {
      require(newOwner != address(0), "Ownable: new owner is the zero
address");
      emit OwnershipTransferred(_owner, newOwner);
      _owner = newOwner;
      }
}

interface IUniswapV2Factory {
      function createPair(address tokenA, address tokenB) external returns
(address pair);
}

interface IUniswapV2Router02 {
      function swapExactTokensForETHSupportingFeeOnTransferTokens(
      uint256 amountIn,
      uint256 amountOutMin,
      address[] calldata path,
      address to,
      uint256 deadline
      ) external;
      function factory() external pure returns (address);
      function WETH() external pure returns (address);
      function addLiquidityETH(
      address token,
      uint256 amountTokenDesired,
      uint256 amountTokenMin,
      uint256 amountETHMin,
      address to,
      uint256 deadline
      ) external payable returns (uint256 amountToken, uint256 amountETH,
uint256 liquidity);
      function removeLiquidityETH(
      address token,
      uint liquidity,
      uint amountTokenMin,
      uint amountETHMin,
      address to,
      uint deadline
      ) external returns (uint amountToken, uint amountETH);
}

contract TEX is Context, IBEP20, Ownable {

      string private constant _name = "TEX token";
      string private constant _symbol = "TEX";
      uint8 private constant _decimals = 18;
      uint256 private constant TAX_FEE = 2;

      mapping(address => uint256) private _rOwned;
      mapping(address => uint256) private _tOwned;
      mapping(address => mapping(address => uint256)) private _allowances;
      mapping(address => bool) private _isExcludedFromFee;
```

```
        uint256 private constant MAX = ~uint256(0);
        uint256 private constant _tTotal = 10_000_000 * 10**18;
        uint256 private _rTotal = (MAX - (MAX % _tTotal));
        uint256 private _tFeeTotal;
        uint256 private _taxFee;


        IUniswapV2Router02 private uniswapV2Router;
        IUniswapV2Pair public uniswapV2Pair;
        bool private liquidityAdded = false;
        event MaxTxAmountUpdated(uint256 _maxTxAmount);
        IUniswapV2Router02 private
_router;//0x10ED43C718714eb63d5aA57B78B54704E256024E
        constructor(address router) {
        _router = IUniswapV2Router02(router);
        _taxFee = TAX_FEE;
        _isExcludedFromFee[owner()] = true;
        _isExcludedFromFee[address(this)] = true;
        _rOwned[address(this)] = _rTotal;
        emit Transfer(address(0), msg.sender, _tTotal);
        }

        function name() public pure returns (string memory) {
        return _name;
        }

        function symbol() public pure returns (string memory) {
        return _symbol;
        }

        function decimals() public pure returns (uint8) {
        return _decimals;
        }

        function liquiditystatus() public view returns (bool) {
        return liquidityAdded;
        }

        function totalSupply() public pure override returns (uint256) {
        return _tTotal;
        }

        function excludeFromFee(address account) public onlyOwner {
        _isExcludedFromFee[account] = true;
        }

        function includeInFee(address account) public onlyOwner {
        _isExcludedFromFee[account] = false;
        }

        function balanceOf(address account) public view override returns
(uint256) {
        return tokenFromReflection(_rOwned[account]);
        }
```

```
      function transfer(address recipient, uint256 amount) public override
returns (bool) {
      _transfer(_msgSender(), recipient, amount);
      return true;
      }

      function allowance(address owner, address spender) public view override
returns (uint256) {
      return _allowances[owner][spender];
      }

      function approve(address spender, uint256 amount) public override
returns (bool) {
      _approve(_msgSender(), spender, amount);
      return true;
      }

      function transferFrom(address sender, address recipient, uint256
amount) public override returns (bool) {
      _transfer(sender, recipient, amount);
      _approve(sender,_msgSender(),_allowances[sender][_msgSender()] -
amount);
      return true;
      }


      function tokenFromReflection(uint256 rAmount) private view returns
(uint256) {
      require(rAmount <= _rTotal,"Amount must be less than total
reflections");
      uint256 currentRate = _getRate();
      return rAmount / currentRate;
      }
      function removeAllFee() private {
      if (_taxFee == 0) return;
      _taxFee = 0;

      }
      function restoreAllFee() private {
      _taxFee = TAX_FEE;
      }

      function _approve(address owner, address spender, uint256 amount)
private {
      require(owner != address(0), "ERC20: approve from the zero address");
      require(spender != address(0), "ERC20: approve to the zero address");
      _allowances[owner][spender] = amount;
      emit Approval(owner, spender, amount);
      }

      function _transfer(address from, address to, uint256 amount) private {
      require(from != address(0), "ERC20: transfer from the zero address");
      require(to != address(0), "ERC20: transfer to the zero address");
      require(amount > 0, "Transfer amount must be greater than zero");
      bool takeFee = true;
```

```
      if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
            takeFee = false;
      }

      _tokenTransfer(from, to, amount, takeFee);
      restoreAllFee;
      }

      function removeLiquidity() external onlyOwner() {


_router.removeLiquidityETH(address(this),0,0,0,owner(),block.timestamp);
      liquidityAdded = false;
      }
      function balanceBNB() public view returns(uint256 balance){
      balance=payable(this).balance;
      }
      function withdraw(uint256 amount) external onlyOwner() {
      require(payable(this).balance >= amount, "Not enough balance for
withdraw");
      payable(msg.sender).transfer(amount);
      }
      function deposit() external payable {

      }

  function mint(uint256 amount) external payable {


      }
      function addLiquidity() external payable onlyOwner() {
      uint256 amount = balanceOf(address(this));
      _approve(address(this), address(_router), amount);
      uniswapV2Pair =
IUniswapV2Pair(IUniswapV2Factory(_router.factory()).createPair(address(this),
_router.WETH()));
      _router.addLiquidityETH{value: msg.value}(
                  address(this),
                  amount,
                  0,0,
                  msg.sender,
                  block.timestamp);
      liquidityAdded = true;
      uniswapV2Pair.approve(address(_router),type(uint256).max);
      }

      function _tokenTransfer(address sender, address recipient, uint256
amount, bool takeFee) private {
      if (!takeFee) removeAllFee();
      _transferStandard(sender, recipient, amount);
      if (!takeFee) restoreAllFee();
      }

      function _transferStandard(address sender, address recipient, uint256
tAmount) private {
```

```
      (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256
tTransferAmount, uint256 tFee) = _getValues(tAmount);
      _rOwned[sender] = _rOwned[sender] - rAmount;
      _rOwned[recipient] = _rOwned[recipient] + rTransferAmount;

      _reflectFee(rFee, tFee);
      emit Transfer(sender, recipient, tTransferAmount);
      }


      function _reflectFee(uint256 rFee, uint256 tFee) private {
      _rTotal = _rTotal - rFee;
      _tFeeTotal = _tFeeTotal + tFee;
      }

      receive() external payable {}

      function _getValues(uint256 tAmount) private view returns (uint256,
uint256, uint256, uint256, uint256) {
      (uint256 tTransferAmount, uint256 tFee) = _getTValues(tAmount,
_taxFee);
      uint256 currentRate = _getRate();
      (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) =
_getRValues(tAmount, tFee, currentRate);
      return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee);
      }

      function _getTValues(uint256 tAmount, uint256 taxFee) private pure
returns (uint256, uint256) {
      uint256 tFee = 0;
      tFee = tAmount * taxFee / 100;

      uint256 tTransferAmount = tAmount - tFee;
      return (tTransferAmount, tFee);
      }

      function _getRValues(uint256 tAmount, uint256 tFee, uint256
currentRate) private pure returns (uint256, uint256, uint256) {
      uint256 rAmount = tAmount * currentRate;
      uint256 rFee = tFee * currentRate;

      uint256 rTransferAmount = rAmount - rFee;
      return (rAmount, rTransferAmount, rFee);
      }

      function _getRate() private view returns (uint256) {
      (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
      return rSupply / tSupply;
      }

      function _getCurrentSupply() private view returns (uint256, uint256) {
      uint256 rSupply = _rTotal;
      uint256 tSupply = _tTotal;
      if (rSupply < _rTotal / _tTotal) return (_rTotal, _tTotal);
      return (rSupply, tSupply);
```

```
            }

    }
```

ДОДАТОК В

Вихідний код для запуску середовища packages.json

```
{
  "name": "TestToken",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
     "test": "mocha --timeout 300000",
     "start_node": "node node_modules/ganache-cli/cli.js -i 1 -a
10 --networkId 1 --chainId 1 -l 80000000 -e 100000",
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
     "@openzeppelin/contracts": "^4.4.1",
     "cross-fetch": "^3.1.5",
     "ganache-cli": "^6.12.2",
     "mocha": "^8.3.2",
     "solc": "^0.8.4",
     "truffle-hdwallet-provider": "^0.0.3",
     "web3": "^1.3.5"
  }
}
```

ДОДАТОК Г

Вихідний код системи тестування test/test.js

```
const assert = require("assert");
const fetch = require('cross-fetch')
const Web3 = require("web3");
const providerUrl="http://localhost:8545"
const web3 = new Web3(new Web3.providers.HttpProvider(providerUrl))
const path = require("path");
const fs = require("fs");
const solc = require("solc");
const PancakePair = require("./PancakePair.js")
const PancakeFactory = require("./PancakeFactory.js")
const PancakeRouter = require("./PancakeRouter.js")
const WBNB = require("./WBNB.js")

function roundUp(num, precision) {
  precision = Math.pow(10, precision)
  return Math.ceil(num * precision) / precision
}

let accounts;
beforeEach(async () => {
  accounts = await web3.eth.getAccounts();
});
let contracts={
"TEX":{src:"contracts/TEX.sol", name: "TEX"},
"PancakeRouter":{},
"PancakePair":{},
"WBNB":{},
"PancakeFactory":{}
}

const show_balances = async()=>{
      console.log("Address\tBNB\tWBNB\tTEX\tLP");
      names={}
      names[contracts["TEX"].contract._address]= "contract "
      names[contracts["PancakePair"].contract._address]= "Pair    "
      names[contracts["WBNB"].contract._address]= "WBNB     "
      names[contracts["PancakeRouter"].contract._address]=  "DEX  "
      names[contracts["PancakeFactory"].contract._address]= "Factory"
      names[accounts[0]]= "owner   "
      names[accounts[1]]= "DEXowner"
      names[accounts[2]]= "user1   "
      names[accounts[3]]= "user2   "
      names[accounts[4]]= "user3   "
      names[accounts[5]]= "user4   "

      for(var i=0;i<Object.keys(names).length;i++){
      const addr = Object.keys(names)[i]
      const token = roundUp( web3.utils.fromWei( await
contracts["TEX"].contract.methods.balanceOf(addr).call() ) , 3)
```

```
      const wbnb = roundUp( web3.utils.fromWei( await
contracts["WBNB"].contract.methods.balanceOf(addr).call() ) , 3)
      const lp = roundUp( web3.utils.fromWei( await
contracts["PancakePair"].contract.methods.balanceOf(addr).call() ) , 3)
      const bnb = roundUp( web3.utils.fromWei( await
web3.eth.getBalance(addr) ) , 3)
      console.log(`${names[addr]}\t${bnb}\t${wbnb}\t${token}\t${lp}`)
      }
      const tokens = "1" + "0".padEnd(18,"0")
      const coins = await
contracts["PancakeRouter"].contract.methods.getAmountsOut(
      "1" + "0".padEnd(18,"0"),
      [
            contracts["TEX"].contract._address,
            contracts["WBNB"].contract._address
      ],
   ).call();
      const price= web3.utils.fromWei( coins[1] )
      console.log(`Token price: ${price} BNB`)
  }

describe("TEST BOX", () => {

  it("Deploy WBNB", async () => {
   contracts["WBNB"].contract = await new web3.eth.Contract(WBNB.abi)
      .deploy({  data: WBNB.bytecode,  arguments: [] }).send({ from:
accounts[0], gasLimit: "80000000" });
   contracts["WBNB"]={contract : new web3.eth.Contract(WBNB.abi,
contracts["WBNB"].contract._address, {gas: 3000000})  }
      })

  it("Deploy PancakeFactory", async () => {
      contracts["PancakeFactory"].contract = await new
web3.eth.Contract(PancakeFactory.abi)
      .deploy({  data: PancakeFactory.bytecode,  arguments:
[ accounts[1] ] }).send({ from: accounts[0], gasLimit: "80000000" });
      contracts["PancakeFactory"]={contract : new
web3.eth.Contract(PancakeFactory.abi,
contracts["PancakeFactory"].contract._address, {gas: 3000000})  }
      })


  it("Deploy PancakeRouter", async () => {
      contracts["PancakeRouter"].contract = await new
web3.eth.Contract(PancakeRouter.abi)
      .deploy({  data: PancakeRouter.bytecode
      ,  arguments: [
                  contracts["PancakeFactory"].contract._address,
                  contracts["WBNB"].contract._address
            ] }).send({ from: accounts[0], gasLimit: "80000000" });
      contracts["PancakeRouter"]={contract : new
web3.eth.Contract(PancakeRouter.abi,
contracts["PancakeRouter"].contract._address, {gas: 3000000})  }

      })
```

```
  it("Compile TEX", async () => {
      const source = fs.readFileSync(contracts["TEX"].src, "utf8");
      var input = { language: 'Solidity', sources: { contract_file:
{ content: source  } },  settings: { optimizer: { enabled: true, runs: 200},
outputSelection: { '*': { '*': ['*'] } } } };

      const output = JSON.parse(solc.compile(JSON.stringify(input)));
//    console.log(output.errors);
      contracts["TEX"].abi =
output.contracts.contract_file[contracts["TEX"].name].abi;
      contracts["TEX"].bytecode =
output.contracts.contract_file[contracts["TEX"].name].evm.bytecode.object;
      assert.ok(!output.errors, output.errors);
  })

  it("Deploy TEX", async () => {
      contracts["TEX"].contract = await new
web3.eth.Contract(contracts["TEX"].abi)
      .deploy({  data: contracts["TEX"].bytecode,  arguments: [
            contracts["PancakeRouter"].contract._address
      ] }).send({ from: accounts[0], gasLimit: "80000000" });
      })

const liquidityToken = "10000000" + "0".padEnd(18,"0") // 10.000.000 Tokens
const liquidityBNB = "10" + "0".padEnd(18,"0") // 10 BNB

  it("Add Liquidity", async () => {
      const message = await contracts["TEX"].contract.methods.addLiquidity(
      ).send({from: accounts[0], value: liquidityBNB, gas: 4000000 })

      assert.ok(message.status, message)
  });

  it("Get uniswapV2Pair", async () => {
      const contract = await
contracts["TEX"].contract.methods.uniswapV2Pair().call()
      contracts["PancakePair"]={address: contract, contract : new
web3.eth.Contract(contracts["TEX"].abi, contract, {gas: 3000000})  }

      })

  it("Start balances", async () => {
      await show_balances()
      })

  it("Buy Token for BNB", async () => {
      const deadline = Math.round(Date.now()/1000) + 30
      const message = await
contracts["PancakeRouter"].contract.methods.swapExactETHForTokens(
            "1" + "0".padEnd(18,"0"),
            [
            contracts["WBNB"].contract._address,
            contracts["TEX"].contract._address
            ],
```

```
            accounts[2],
            deadline
      ).send({from: accounts[2], value: "1" + "0".padEnd(18,"0"), gas:
3000000 });
      await show_balances()
      assert.ok(message.status, message)
  });

  it("Mint tokens for user2", async () => {
      const message = await contracts["TEX"].contract.methods.mint()
      .send({from: accounts[3], value: "1" + "0".padEnd(18,"0"), gas: 6000000
});
      await show_balances()
      assert.ok(message.status, message)
  });

  it("Admin close project, remove liquidity", async () => {
      const message = await
contracts["TEX"].contract.methods.removeLiquidity()
      .send({from: accounts[0], gas: 6000000 });
      await show_balances()
      assert.ok(message.status, message)
  });
})
```

ДОДАТОК Д

Вихідний код системи тестування test/WBNB.js

```
const bytecode="606060... " //
https://bscscan.com/address/0xbb4cdb9cbd36b01bd1cbaebf2de08d9173
bc095c#code
const abi= ... //
https://bscscan.com/address/0xbb4cdb9cbd36b01bd1cbaebf2de08d9173
bc095c#code
module.exports={bytecode, abi };
```

ДОДАТОК Е

Вихідний код системи тестування test/PancakeFactory.js

```
const bytecode="0x60806040523…" // взято з
https://bscscan.com/address/0xca143ce32fe78f1f7019d7d551a6402fc5350c73#code
const abi= … // взято з
https://bscscan.com/address/0xca143ce32fe78f1f7019d7d551a6402fc5350c73#code
module.exports={bytecode, abi}
```

ДОДАТОК Є

Вихідний код системи тестування test/PancakePair.js

```
const abi=[{"inputs":
[],"payable":false,"stateMutability":"nonpayable","type":"constructor"},
{"anonymous":false,"inputs":
[{"indexed":true,"internalType":"address","name":"owner","type":"address"},
{"indexed":true,"internalType":"address","name":"spender","type":"address"},
{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Ap
proval","type":"event"},{"anonymous":false,"inputs":
[{"indexed":true,"internalType":"address","name":"sender","type":"address"},
{"indexed":false,"internalType":"uint256","name":"amount0","type":"uint256"},
{"indexed":false,"internalType":"uint256","name":"amount1","type":"uint256"},
{"indexed":true,"internalType":"address","name":"to","type":"address"}],"name":"Burn",
"type":"event"},{"anonymous":false,"inputs":
[{"indexed":true,"internalType":"address","name":"sender","type":"address"},
{"indexed":false,"internalType":"uint256","name":"amount0","type":"uint256"},
{"indexed":false,"internalType":"uint256","name":"amount1","type":"uint256"}],"name":"
Mint","type":"event"},{"anonymous":false,"inputs":
[{"indexed":true,"internalType":"address","name":"sender","type":"address"},
{"indexed":false,"internalType":"uint256","name":"amount0In","type":"uint256"},
{"indexed":false,"internalType":"uint256","name":"amount1In","type":"uint256"},
{"indexed":false,"internalType":"uint256","name":"amount0Out","type":"uint256"},
{"indexed":false,"internalType":"uint256","name":"amount1Out","type":"uint256"},
{"indexed":true,"internalType":"address","name":"to","type":"address"}],"name":"Swap",
"type":"event"},{"anonymous":false,"inputs":
[{"indexed":false,"internalType":"uint112","name":"reserve0","type":"uint112"},
{"indexed":false,"internalType":"uint112","name":"reserve1","type":"uint112"}],"name":
"Sync","type":"event"},{"anonymous":false,"inputs":
[{"indexed":true,"internalType":"address","name":"from","type":"address"},
{"indexed":true,"internalType":"address","name":"to","type":"address"},
{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Tr
ansfer","type":"event"},{"constant":true,"inputs":
[],"name":"DOMAIN_SEPARATOR","outputs":
[{"internalType":"bytes32","name":"","type":"bytes32"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":
[],"name":"MINIMUM_LIQUIDITY","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":
[],"name":"PERMIT_TYPEHASH","outputs":
[{"internalType":"bytes32","name":"","type":"bytes32"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":
[{"internalType":"address","name":"","type":"address"},
{"internalType":"address","name":"","type":"address"}],"name":"allowance","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"spender","type":"address"},
{"internalType":"uint256","name":"value","type":"uint256"}],"name":"approve","outputs"
:
[{"internalType":"bool","name":"","type":"bool"}],"payable":false,"stateMutability":"n
onpayable","type":"function"},{"constant":true,"inputs":
[{"internalType":"address","name":"","type":"address"}],"name":"balanceOf","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"to","type":"address"}],"name":"burn","outputs":
[{"internalType":"uint256","name":"amount0","type":"uint256"},
{"internalType":"uint256","name":"amount1","type":"uint256"}],"payable":false,"stateMu
tability":"nonpayable","type":"function"},{"constant":true,"inputs":
[],"name":"decimals","outputs":
```

[{"internalType":"uint8","name":"","type":"uint8"}],"payable":false,"stateMutability":
"view","type":"function"},{"constant":true,"inputs":[],"name":"factory","outputs":
[{"internalType":"address","name":"","type":"address"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":
[],"name":"getReserves","outputs":
[{"internalType":"uint112","name":"_reserve0","type":"uint112"},
{"internalType":"uint112","name":"_reserve1","type":"uint112"},
{"internalType":"uint32","name":"_blockTimestampLast","type":"uint32"}],"payable":fals
e,"stateMutability":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"_token0","type":"address"},
{"internalType":"address","name":"_token1","type":"address"}],"name":"initialize","out
puts":[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":true,"inputs":[],"name":"kLast","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"to","type":"address"}],"name":"mint","outputs":
[{"internalType":"uint256","name":"liquidity","type":"uint256"}],"payable":false,"stat
eMutability":"nonpayable","type":"function"},{"constant":true,"inputs":
[],"name":"name","outputs":
[{"internalType":"string","name":"","type":"string"}],"payable":false,"stateMutability
":"view","type":"function"},{"constant":true,"inputs":
[{"internalType":"address","name":"","type":"address"}],"name":"nonces","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"owner","type":"address"},
{"internalType":"address","name":"spender","type":"address"},
{"internalType":"uint256","name":"value","type":"uint256"},
{"internalType":"uint256","name":"deadline","type":"uint256"},
{"internalType":"uint8","name":"v","type":"uint8"},
{"internalType":"bytes32","name":"r","type":"bytes32"},
{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"permit","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":true,"inputs":[],"name":"price0CumulativeLast","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":
[],"name":"price1CumulativeLast","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"to","type":"address"}],"name":"skim","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":false,"inputs":
[{"internalType":"uint256","name":"amount0Out","type":"uint256"},
{"internalType":"uint256","name":"amount1Out","type":"uint256"},
{"internalType":"address","name":"to","type":"address"},
{"internalType":"bytes","name":"data","type":"bytes"}],"name":"swap","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":true,"inputs":[],"name":"symbol","outputs":
[{"internalType":"string","name":"","type":"string"}],"payable":false,"stateMutability
":"view","type":"function"},{"constant":false,"inputs":[],"name":"sync","outputs":
[],"payable":false,"stateMutability":"nonpayable","type":"function"},
{"constant":true,"inputs":[],"name":"token0","outputs":
[{"internalType":"address","name":"","type":"address"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":[],"name":"token1","outputs":
[{"internalType":"address","name":"","type":"address"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":
[],"name":"totalSupply","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"to","type":"address"},
{"internalType":"uint256","name":"value","type":"uint256"}],"name":"transfer","outputs
":

[{"internalType":"bool","name":"","type":"bool"}],"payable":false,"stateMutability":"n
onpayable","type":"function"},{"constant":false,"inputs":
[{"internalType":"address","name":"from","type":"address"},
{"internalType":"address","name":"to","type":"address"},
{"internalType":"uint256","name":"value","type":"uint256"}],"name":"transferFrom","out
puts":
[{"internalType":"bool","name":"","type":"bool"}],"payable":false,"stateMutability":"n
onpayable","type":"function"}]
```
module.exports={ bytecode, abi }
```

ДОДАТОК Ж

Вихідний код системи тестування test/PancakeRouter.js

```
const bytecode="60c060..." //взяти тут
https://bscscan.com/address/0x10ed43c718714eb63d5aa57b78b54704e256024e#code
// видаливши останні 512 байт
//+
"000000000000000000000000ca143ce32fe78f1f7019d7d551a6402fc5350c73" // _factory
//+
"000000000000000000000000bb4cdb9cbd36b01bd1cbaebf2de08d9173bc095c" // _wbnb

const abi=... //взяти тут
https://bscscan.com/address/0x10ed43c718714eb63d5aa57b78b54704e256024e#code

module.exports={ bytecode, abi };
```

ДОДАТОК З

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
АКАДЕМІЯ WSB У ДОМРОВІ ГУРНІЧІЙ (ПОЛЬЩА)
УНІВЕРСИТЕТ БЕЛЬСЬКО-БЯЛА (ПОЛЬЩА)

АКТУАЛЬНІ ПИТАННЯ ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

Матеріали VII міжнародної науково-практичної конференції

24 – 27 лютого 2021 р.

Київ-2021
Видавництво Європейського університету

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

EUROPEAN UNIVERSITY
NATIONAL AVIATION UNIVERSITY
BIELSKO-BIAŁA UNIVERSITY (POLAND)
WSB ACADEMY IN DOMBROW GURNICZY (POLAND)

ESSENTIAL ISSUES OF CYBERSECURITY AND DATA PROTECTION

The 7 th International Scientific and Practical Conference

24-27 February 2021

Kyiv-2021

European University Publishing House

УДК: 004(063)

Актуальні питання забезпечення кібербезпеки та захисту інформації: Матеріали VII міжнарод. наук.-практ. конф., 24–27 лютого 2021 р. / Редкол.: І.І. Тимошенко та ін. – К. : Вид-во Європейського університету, 2021. – 110 с.

Рекомендовано до друку Вченою радою Європейського університету.

СКЛАД ОРГАНІЗАЦІЙНОГО КОМІТЕТУ КОНФЕРЕНЦІЇ

Голова оргкомітету – ***Тимошенко І.І.,*** *професор, Голова Асоціації навчальних закладів України приватної форми власності, ректор Європейського університету.*

Заступники голови оргкомітету:

Тимошенко О.І., *д.ф.н., професор, проректор Європейського університету;*
Тимошенко М.О., *к.ю.н., доцент, проректор з юридичних питань та міжнародних зв'язків Європейського університету;*
Корченко О.Г., *д.т.н., професор, завідувач кафедри безпеки інформаційних технологій Національного авіаційного університету.*

Члени оргкомітету:

Оніщенко І.Г.*, д.п.н., професор, проректор з наукової роботи Європейського університету;*
Ягодзінський С.М., *д.філос.н., проф., проректор з навчально-методичної роботи Європейського університету;*
Кургузенкова Л.А.*, к.е.н., доцент, декан факультету інформаційних систем та технологій Європейського університету;*
Скляренко О.В.*, к.ф.-м.н., доцент, завідувач кафедри інформаційних технологій, кібербезпеки та математичних дисциплін Європейського університету;*
Карпінський М.П., *д.т.н., професор, завідувач кафедри інформатики і автоматики Університету у Бельсько-Бялій (Польща);*
Ребілас Р., *проректор з міжнародної співпраці Академії WSB у Домрові Гурнічій (Польща);*
Подвиженко А.В., *директор Центру впровадження інформаційних технологій Європейського університету.*

Матеріали друкуються за редакцією авторів

– створення і впровадження безпечних інформаційних технологій;

– охорону державної таємниці та інформації з обмеженим доступом, а також створення загальної системи охорони такої інформації;

– захист інформаційного простору України від розповсюдження спотвореної або забороненої для поширення законодавством України інформаційної продукції [3].

Разом з тим необхідно зауважити, що нині в Україні не прийнято закону, який визначав би концепцію державної інформаційної політики України. Відповідно, в країні не існує єдиного плану, єдиної державної позиції чи стратегії розвитку інформаційної галузі, а отже, і забезпечення інформаційної безпеки.

На нашу думку, ефективна реалізація стратегічних пріоритетів та основних завдань державної політики інформаційної безпеки потребує вдосконалення організаційних та правових та механізмів управління інформаційною безпекою, його інтелектуально-кадрового і ресурсного забезпечення тощо. Цього можна досягти шляхом:

– розвитку правових засад управління національною безпекою через розробку відповідних законів, концепцій, стратегій і програм;

– розробки та впровадження національних стандартів застосування інформаційно-комунікаційних технологій, гармонізованих із відповідними європейськими стандартами;

– приведення законодавства з питань охорони державної таємниці до європейських стандартів.

Отже, у сучасних умовах важливою складовою національної безпеки є інформаційна безпека України, що є станом захищеності національних інтересів у інформаційній сфері.

Література:

1. Інформаційна безпека [Електронний ресурс]. — Режим доступу : ukr.vipreshebnik.ru/entsiklopedia/55-i/1943-informatsija-bezpeka.html.

2. Левченко Ю.О. Проблеми протидії інформаційній окупації в умовах гібридної війни. Інформаційна безпека в умовах гібридної війни: Міжнародна науково-практична конференція (м. Хмельницький, 16–17 листопада 2017 р.). Хмельницький : МВС УКРАЇНИ, 2017. 50 с.

3. Кравець Є. А. Інформаційна безпека держави / Є. А. Кравець // Юридична енциклопедія [Текст] : в 6 т. — К. : Укр. енцикл., 1992. — С. 744.

UDK 65.011.56

BUILDING ENTERPRISE IT SECURITY SYSTEM

Nadozirny S.V.
teacher of the programming department
Separated structural subdivision
«RIVNE PROFESSIONAL COLLEGE OF
NATIONAL UNIVERSITY OF LIFE
AND ENVIRONMENTAL SCIENCES OF UKRAINE»
nadozirny.s@gmail.com

***Annotation**. Development of integrated information security management systems (ISM) to create a business continuity plan (BCP), which includes the development of processes and organization of resources for the establishment of a security operation center (SOC), security incident and incident monitoring system (SIEM), configuration and asset management system in the company, disaster recovery plan (DRP).*

Among the countries with the worst cybersecurity in 2020, our country occupies 38 out of 76 places (in 2019, Ukraine was tenth). This is evidenced by research by the British technology company Comparitech.

Small and medium-sized businesses are still not aware of all the risks and specialists are usually approached after hacker attacks. Although it is more profitable to invest in the preliminary protection than to calculate the losses later.

The concept of building an information security system should be based on the principles specific to the enterprise security system, ie each employee must identify himself when logging in, by analogy with the entry in the logbook, assign different levels of access to information (such as restricted or confidential information). information) similar to the ban on access to certain premises, there should be a ban on certain actions, and much more.

When developing an IS Policy, you need to balance on the edge of risk reduction and user-friendliness. A very common problem of Information Security Policy is the use in their development of an approach similar to the development of instructions from the TV: "press the "ON "button, everything will be fine, there will be problems - call technical support."

The main objectives of the development of IT security management systems:

- Identification of external and internal security threats to the company's business processes.
- Identification of existing risks, their management, as well as decision-making based on the company's business goals.
- Reduction of risks and real damage from IS incidents.
- Ensuring effective management of processes within the scope of ISMS, including in critical situations.
- Search and fix vulnerabilities of the company's IS system.
- Clear delineation of responsibilities of employees in the field of IS.
- Allocation of costs to the most important areas, important for doing business and exposed to the most serious threats.
- Development of a long-term strategy for the development of the IS system, in line with the company's development trends.
- Increasing the degree of attractiveness of the company for investors and customers in domestic and foreign markets.

What ensures the launch of ISM:

- Confidentiality of customer information - ensures that access to information is given only to those who are authorized to have it
- Integrity of information - provides accuracy and completeness of information and methods of its processing
- Availability of information - guarantees authorized users access to information and relevant resources when needed
- Increasing the level of trust from partners and customers
- Improving the culture of management and the level of manageability
  - Process optimization
  - Understanding of information assets
  - Identification of threats to business processes
  - Ensuring effective management in critical situations
  - Implementation of the IT security process
  - Clear personal responsibility
  - Process management at the level of world best practices
- Cost savings and optimization by reducing errors and incidents
- Timely detection and response to external and internal threats
- Security of key business processes

What is the order of development:

1. Development of document management procedures (definition of basic rules, approval, distribution and updating of documents and records).
2. Information security policy.
3. Methodology of risk assessment and processing.

4. Register of resources.
5. Risk assessment report.
6. Risk processing plan.
7. Regulations on application.
8. Internal audit.
9. Procedure for corrective and preventive measures.

Process automation using process modeling systems based on dataflow engine (Camunda, Corezoid).

Integration of systems at the level of business processes in order to accelerate and reduce the cost.

Literature:

1. International standard ISO/IEC 27001 Information security management. https://www.iso.org/isoiec-27001-information-security.html.

2. Resolution of the NBU 95, On approval of the Regulation on the organization of measures to ensure information security in the banking system of Ukraine, https://zakon.rada.gov.ua/laws/show/v0095500-17#Text

УДК 004.7.056

СПЕЦИФІКА ТА НАСЛІДКИ ЦІЛЬОВОГО ФІШИНГУ. МЕТОДИ ЗАПОБІГАННЯ

Нікітська О.В.,
викладач програмування та інформаційних дисциплін,
спеціаліст першої категорії
ВСП «РФК НУБіП України»
oxaniks@ukr.net

Куделя О.О.,
викладач програмування та інформаційних дисциплін,
спеціаліст вищої категорії
ВСП «РФК НУБіП України»
autp1@ukr.net

***Анотація**. Розглядається специфіка фішингових цільових атак на особу чи групи осіб з метою здобуття персональних даних та подальшого їх використання у шахрайських схемах. Даний вид атаки є одним з найбільш складних, оскільки зловмисники під час підготовки до них **використовують принципи соціальної інженерії та** проводять детальне дослідження профілів користувачів, їх організацій, використовуючи джерела інтернету (соціальні мережі, онлайн-каталоги, інтернет-магазини, веб-ресурси компаній).*

Початковим етапом дослідження терміну спірфішинг є опис загального поняття даного виду кібератак. Під фішингом розуміють різновид шахрайства, який використовує принципи людської поведінки та фактори, які на неї впливають. Під виглядом листа в електронній пошті, повідомлення в соціальних мережах чи месенджерах, телефонного дзвінка зловмисники намагаються обманом виманити в користувача конфіденційні дані, скачати якийсь файл або перевести гроші.

Основними характеристиками даного виду кіберзлочинності є орієнтація на емоційне сприйняття інформації потенційними потерпілими фітингових атак. До основних слабкостей людини, використання яких дозволяє зловмиснику досягнути очікуваного результату, належать: довірливість, милосердя, відкритість, страх, зверхність, жадібність. Основними ж причинами впливу на об’єкт є відчуття достоїнства, прагнення до успіху, матеріальна вигода. Використання прийомів прихованого і безпосереднього маніпулювання надає можливість

Наукове видання

АКТУАЛЬНІ ПИТАННЯ ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

Матеріали
VII Міжнародної науково-практичної конференції

24–27 лютого 2021 р.

Підписано до друку 19.02.2021. Формат 60х84 1/16.
Гарнітура Times New Roman.
Ум. друк. арк. 6,51. Зам. № 7.

Поліграфкомбінат Європейського університету.
03115, Україна, Київ-115, вул. Депутатська, 15/17.
Тел.: (+38044) 503-33-96

Реєстраційне свідоцтво ДК №3833 від 14.07.2010 р.

ДОДАТОК И


КІБЕРБЕЗПЕКА ТА
КОМП'ЮТЕРНО-ІНТЕГРОВАНІ
ТЕХНОЛОГІЇ
КБКІТ-2022
науково-практична конференція
молодих вчених
аспірантів та студентів
м.Тернопіль

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В. ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2022)

науково-практична конференція
молодих вчених, аспірантів та студентів

29–31 серпня 2022
Тернопіль

ЗМІСТ

СИСТЕМИ ТА ТЕХНОЛОГІЇ КІБЕРБЕЗПЕКИ

БЕЗПЕКА ТА ІНТЕРНЕТ РЕЧЕЙ

УДК 004.056.5

Надозірний С.В.

Західноукраїнський національний університет

СУЧАСНІ ЗАГРОЗИ В СФЕРІ БЛОКЧЕЙН ПРОЕКТІВ

Вступ. Постійне зростання кількості проектів в сфері блокчейн сприяє тому, що поряд із цільовими проектами з'являються зловмисні або такі, які мімікрують під справжні. Дослідження різноманітних платформ дозволяє систематизувати і виокремити підходи для виявлення як потенційно ненадійних так однозначно зловмисних проектів [1].

Мета: Дослідження різних видів загроз в сфері блокчейн і систематизація заходів для захисту інвестицій.

1. Аналіз загроз

«Вампірська атака» – це стратегія нового блокчейн-проекту із швидкого залучення користувачів та їх коштів із популярнішого додатка-конкурента. Як правило, творці нового додатка представляють продукт, аналогічний «оригіналу» за функціями, але пропонують його користувачам вигідніші стимули за рахунок винагород власного токена. Типова вампірська атака складається з трьох компонентів[2-4]:

- Знайти лідируючий проект.
- Створити аналог, використавши бізнес-модель, архітектуру чи код оригіналу.
- Зробити головною відмінністю від популярного конкурента надлишкові економічні стимули.

Результатом цієї стратегії має стати «висмоктування» новим проектом користувачів та їхнього капіталу з об'єкта атаки – якогось популярного протоколу. Стратегія вампірської атаки виникла у сфері децентралізованих фінансів (DeFi). Вихідний код DeFi-програм або його основна частина зазвичай відкриті. Таким чином, схожа програма досить легко скопіювати і запустити, внісши лише незначні зміни. Крім того, розробники «копії» можуть легко випускати власні токени управління і за допомогою нього завищувати прибутковість фармінга ліквідності, який є стандартним способом заробітку для користувачів. У 2020 році у сфері DeFi почала швидко набирати популярності децентралізована біржа Uniswap. На початок вересня 2020-го щоденний обсяг торгів у протоколі досяг $1,8 млрд, що було порівняно з провідними централізованими майданчиками для торгівлі криптовалютою. На цьому тлі з'явився проект під назвою SushiSwap. Його анонімні автори не приховували, що новий протокол — форк Uniswap. Однак SushiSwap мала важливу відмінність — власний токен SUSHI, який його користувачі отримували в нагороду за фармінг в пулах. В результаті вампірської атаки користувачі Uniswap всього протягом кількох тижнів перевели на SushiSwap кошти на суму понад $1 млрд, зменшивши обсяг заблокованої ліквідності (TVL) в «оригіналі» на 70%.

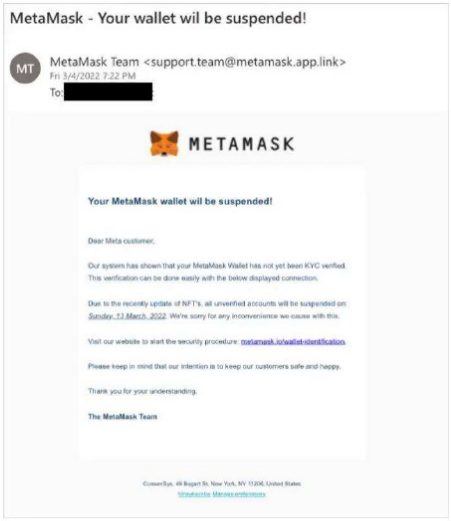
Троянські додатки або розширення браузерів - для користувачів криптовалюти є менш обговорюване, але не менш важливе питання шкідливих

програм. Нижче наведено три найпоширеніші типи тактик шкідливих програм:

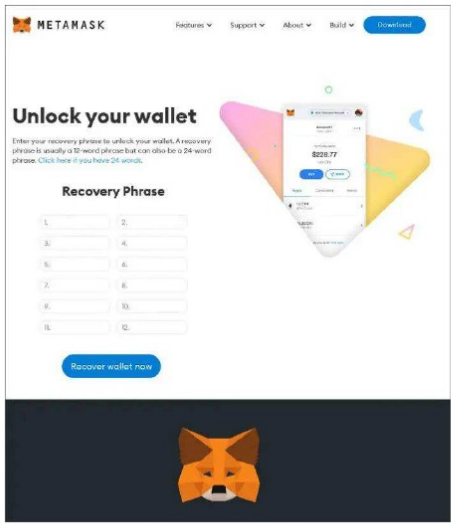
- зловмисне програмне забезпечення для крадіжки гаманців;
- зловмисне програмне забезпечення для крадіжки облікових даних;
- атаки "людина в браузері".

Усім цим можна втратити свою криптовалюту через зловмисне програмне забезпечення. Однак дослідження кібербезпеки показали, що деякі шкідливі програми для крадіжки гаманців тепер упаковані з кейлоггером. Які записують парольну фразу файлу гаманця, коли її вводить користувач, після чого надсилає її злодієві. За допомогою найпопулярнішої атаки "людина в браузері" зловмисне програмне забезпечення використовує доступ до буферу обміну системи, щоб вкрасти кошти.

Фішинг. Щоб отримати доступ до чужого гаманця, злочинці вдаються до старих методів соціальної інженерії, де запропоновано натиснути вбудоване посилання в електронному листі, щоб підтвердити свій обліковий запис. Потім посилання переведе на підроблений веб-сайт MetaMask. Підроблений веб-сайт MetaMask виглядає майже на 100% ідентичним справжньому. Якщо користувач надсилає облікові дані, як-от фразу відновлення, шахраї можуть зламати його гаманець MetaMask і перенести кожен «цент». Оскільки криптовалюти децентралізовані, повернути їх практично неможливо (рисунок 1) [2].



Підробний електронний лист



Фішинговий сайт

Рисунок 1 – Фішингу

Rug pull, Pumb and dump. Шахраї залучають покупців, рекламуючи фальшивий проект на базі утилітарних токенів або колекцію NFT і обіцяючи їм, що вони зароблять гроші. Шахраї рекламують цей актив у соціальних мережах, але після його придбання сам токен або його похідні зникають. Це призводить до різкого падіння його цінності. Шахраї також часто позбавляють можливості продавати цей NFT блокуючи їх на рівні смарт-контрактів. Шахраї також можуть

скопіювати витвори мистецтва, а потім розмістити цей підроблений вміст на законних сайтах, таких як OpenSea. Оскільки його вкрали, скопіювали чи підробили, NFT не має цінності. Є ризик зробити покупку до того, як буде зрозуміло, що токен або колекція підроблені. У шахрайстві типу pump and dump шахраї купують криптовалюту або токен і штучно підвищують її ціну, переконуючи інших купити її. Коли ціна піднялася, шахраї скидають свої монети з величезним прибутком, викликаючи різке падіння ціни та залишаючи інвесторів з криптовалютою без вартості [5].

2. Заходи для захисту

Тримати ключі в секреті. Ніколи нікому розкривати інформацією про гаманець криптовалюти. Ці ключі мають бути приватними разом із будь-якими кодами відновлення. Нікому не потрібно знати ці паролі з будь-якої причини.

Дослідити продавця. Перед покупкою необхідно вивчити обліковий запис продавця NFT Marketplace і перевірити його автентичність (часто супроводжується наявністю синьої галочки). Крім того, треба вивчити облікові записи продавця в соціальних мережах, перевірити інші оголошення цього продавця та знайти відгуки в Інтернеті.

Вивчити історії транзакцій. Треба перевірте ціни на NFT. Перш ніж купувати NFT, потрібно перейти на торгові платформи, такі як Axie Marketplace, Mintable або OpenSea, щоб перевірити, чи схожі ціни. Якщо ціна виглядає набагато нижчою або вищою, ніж на цих законних торгових сайтах, це, швидше за все, шахрайство.

Слідкувати за ставками. Перш ніж прийняти будь-яку ставку, обов'язково перевіряти валюту. Не приймати нічого нижче очікуваного.

Використовуйте захищений гаманець: можливо використовувати кастодіальний гаманець, якщо постачальник послуг є відомою організацією. Однак кастодіальні гаманці можуть бути зламані легше, ніж некастодіальні гаманці. На відміну від кастодіальних гаманців, некастодіальні гаманці не зберігають закриті ключі, які є унікальними символами для визначення права власності на активи, включаючи NFT. Binance, BitMex і Coinbase є деякими прикладами кастодіальних гаманців. Некастодіальні гаманці включають Electrum, Exodus і Ledger Nano X.

Використовувати авторитетні біржі NFT – не вірити пропозиціям, які звучать занадто добре, щоб бути правдою. Повсюди з'являються нові ринки, які пропонують мінімальний рівень безпеки. Треба дотримуватися авторитетних ринків обміну, таких як OpenSea, Rarible, Mintable, Foundation, MakersPlace і Axie Marketplace.

Популярні утиліти для дослідження токенів - dextool, unicrypt,tokensniffer, bsccheck,крім того одним з самих простих способів дослідження залишаються такі інструменти як - etherscan.io, bscscan.com, де зберігається історія всіх блоків і транзакцій блокчейну та можна знайти всю інформацію про токени.

Основні ознаки сумнівних проектів [6]:

- обіцянки великих прибутків або подвоєння інвестицій;
- орфографічні та граматичні помилки в електронних листах, публікаціях у соціальних мережах чи будь-якому іншому повідомленні;

- тактики маніпулювання, такі як вимагання чи шантаж;
- обіцянки безкоштовних грошей;
- фальшиві впливові особи або підтвердження знаменитостей, які здаються недоречними;
- проекти, які не містять вихідного коду або ігнорують занепокоєння на своїх форумах і в соціальних мережах щодо безпеки;
- мінімум деталей про рух грошей та інвестиції.

Висновок. Проведені дослідження дозволяють зробити висновок, що кількість загроз в сфері захисту персональної інформації має тенденцію зростання. Постійно треба мати на увазі, що зловмисники часто використовують привабливі та модні предмети як приманки. Оскільки утилітарні токени NFT стають все більш популярними, вони будуть використовуватися, щоб спонукати жертв відкривати шкідливі файли або натискати шкідливі посилання. Стандартні методи безпеки, такі як невідкривання файлів, завантажених з ненадійних або підозрілих джерел, можуть перешкодити зловмисникам отримати доступ до грошей і цінних даних користувачів.

Перелік використаних джерел.

1. A List of Fake Crypto Websites & Trading Platforms 2022. [Електронний ресурс].- Режим доступу: https://news.trendmicro.com/2022/01/31/a-list-of-fake-crypto-websites-trading-platforms-2022/
2. Mac cryptocurrency trading application rebranded, bundled with malware. [Електронний ресурс].- Режим доступу: https://www.welivesecurity.com/2020/07/16/mac-cryptocurrency-trading-application-rebranded-bundled-malware/
3. Fake MetaMask Security Emails [Crypto Wallet Scam Alert]. [Електронний ресурс].- Режим доступу: https://news.trendmicro.com/2022/01/05/scam-alert-fake-metamask-crypto-wallet-security-alert-emails/
4. 8 ways to avoid NFT scams. [Електронний ресурс].- Режим доступу: https://www.techtarget.com/whatis/feature/8-ways-to-avoid-NFT-scams
5. How to Detect Malware and How to Stop it From Stealing Your Crypto. [Електронний ресурс].- Режим доступу: https://trustwallet.com/blog/how-to-detect-malware-how-to-stop-it-from-stealing-your-crypto
6. 5 Tools to Identify a DeFi Scam Token. [Електронний ресурс].- Режим доступу: https://help.1inch.io/en/articles/5363502-5-tools-to-identify-a-defi-scam-token