НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 003.26+004.056.2

«До захисту допущено»
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2022 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Методи забезпечення безпеки використання смарт-контрактів в енергетиці»**

Виконав:
студент II курсу, групи ФІ-12мп
Слуцький Андрій Сергійович _____

Керівник:
д.т.н., проф. кафедри ММЗІ
Ковальчук Людмила Василівна _____

Рецензент:
к.т.н, доц. кафедри ІБ
Стьопочкіна Ірина Валеріївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — другий (магістерський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2022 р.

ЗАВДАННЯ
на магістерську дисертацію

Студент: Слуцький Андрій Сергійович

1. Тема роботи: «*Методи забезпечення безпеки використання смарт-контрактів в енергетиці*», науковий керівник дисертації: д.т.н., проф. кафедри ММЗІ Ковальчук Людмила Василівна,

затверджені наказом по університету №__ від «__» _____ 2022 р.

2. Термін подання студентом роботи: «__» _____ 2022 р.

3. Об'єкт дослідження: *Використання технологій блокчейн в енергетиці*

4. Предмет дослідження: *Використання смарт-контрактів в енергетиці*

5. Перелік завдань: *провести огляд блокчейн мережі, віртуальної машини ефіріум, смарт-контрактів; дослідити канали стану та мереж які взаємодіють з каналами стану; дослідити безпеку каналів стану; розібрати наявні приклади; реалізування смарт-контракта на мові solidity; перевірка смарт-контракта на функціональність.*

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: *Презентація доповіді*

7. Орієнтовний перелік публікацій: *планується доповідь на всеукраїнській конференції*

8. Дата видачі завдання: 10 вересня 2021 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2021 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2021 р.	Виконано
3	Розбір каналів стану та їх реалізації у блокчейні	Січень 2021 р.	Виконано
4	Розбір смарт-контрактів	Травень 2022 р.	Виконано
5	Побудова смарт-контракта	Червень 2022 р.	Виконано
6	Перевірка смарт-контракта у мережі ефіріум	Вересень 2022 р.	Виконано
7	Оформлення дисертації	Грудень 2022 р.	Виконано

Слуцький А. С. _____ Андрій Слуцький

Ковальчук Л. В. _____ Людмила Ковальчук

РЕФЕРАТ

Кваліфікаційна робота містить: 50 стор., 25 джерел.

У цій роботі було досліджено, як використання блокчейн технологій, а саме смарт-контрактів, в енергетиці дозволяє покращити електромережу, а також було дослідженно безпеки смарт-контрактів і державних каналів.

Метою цієї роботи є в тому щоб показати що використання смарт-контрактів і каналів стану може принести користь в електроенергетиці, а можливо й більше у різній інфраструктурі.

Об'єктом дослідження є процес безпечного функціонування смарт-контрактів з використанням блокчейн технологій.

Предметом дослідження є безпечне використання смарт-контрактів у енергетиці.

Як результат було показано смарт-контракт який можна використовувати задля взаємодії з електромережею, а також переваги такого використання.

БЛОКЧЕЙН, СМАРТ-КОНТРАКТИ, ЕНЕРГЕТИКА, КАНАЛИ СТАНУ

ABSTRACT

The qualification work contains: 50 pages, 25 sources.

This paper explored how the use of blockchain technologies, namely smart contracts, in the energy sector can improve the power grid, and also explored the security of smart contracts and public channels.

The purpose of this work is to show that the use of smart contracts and state channels can bring benefits in electric power, and perhaps more so in various infrastructures.

The object of the research is blockchain technologies, smart contracts, state channels.

The subject of the study is the safe use of smart contracts in energy.

As a result, a smart contract that can be used to interact with the power grid, as well as the advantages of such use, was shown.

BLOCKCHAIN, SMART-CONTRACT, ENERGY, STATE CHANNELS

ЗМІСТ

ВСТУП

Актуальність дослідження. Сьогодні тема блокчейну і смарт-контрактів є однією з найбільш популярних і актуальних тем. Діло в тому, що технологія блокчейн є універсальним способом зберігання і обробки інформації. На сьогодні блокчейн може використовуватись як і в маркетингу та комп'ютерних іграх, так і у воєнній сфері та правових установах. Наприклад в Україні у 2017 році із використанням блокчейн технологій було реалізовано оновлену версію інформаційної системи державного земельного кадастру[1].

Вперше ця технологія була описана Стюартом Габером та У. Скоттом Сторнеттою в 1991 році[2], основна ціль була це зберігати документи з неможливістю спотворення чи пошкодження часових позначок на документах. У 1992 році Байер, Габер та Сторнетт використали в проєкт дерево Меркла, що покращило ефективність. Але по справжньому відомою ця технологія стала лише завдяки людині, чи можливо групі людей відомих під псевдонімом Сатоші Накамото у 2008 році[3]. Завдяки Накамото було втілено криптовалюту Bitcoin, де по суті це є книгою обліку для всіх транзакцій в мережі.

З вдосконаленням технології блокчейну в ньому стало можливо використовувати смарт-контракти. Смарт-контракти появились ще у 1996 році людиною яку звати Нік Сабо[4], він хотів реалізувати електронний протокол торгівлі між незнайомими людьми в Інтернеті. Але ефективно це реалізувати лише вдалося з появою Ethereum. Смарт-контракти можна використовувати ефективно в різних правових та державних установах. В цій роботі буде досліджено як можна забезпечити безпеку використання смартконтрактів в інфраструктурі, а також новий підхід використання каналів стану, для більш безпечного і зручної взаємодії з електроенергетикою.

Метою дослідження є забезпечити ефективне та безпечне

використання смарт-контрактів в електроенергетиці при цьому зробивши взаємодію з електромережею простішою, детермінованою та можливою для покращень. Для досягнення цієї мети необхідно розв'язати **задачу дослідження**, яка полягає у реалізації смарт-контракта який буде працювати з каналами стану та буде стійким до основних атак. Для розв'язання задачі необхідно вирішити такі завдання:

1) провести огляд блокчейн мережі, віртуальної машини ефіріум, смарт-контрактів;

2) дослідити канали стану та мереж які взаємодіють з каналами стану;

3) дослідити безпеку каналів стану;

4) розібрати наявні приклади смарт-контрактів;

5) реалізування смарт-контракта на мові solidity;

6) перевірка смарт-контракта на функціональність;

7) перевірка стійкості смарт-контракта на основні атаки.

Об'єктом дослідження є процес безпечного функціонування смарт-контрактів з використанням блокчейн технологій.

Предметом дослідження є безпечне використання смарт-контрактів у енергетиці.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: методи комп'ютерного та статистичного моделювання, програмування, криптографія.

Наукова новизна полягає у новому підході до використання каналів стану при реалізації смарт-контрактів у енергетиці і цей підхід забезпечує дешевший і більш безпечне функціонування смарт-контрактів.

Практичне значення результатів полягає у розробці смарт-контракту, який є безпечним до основних атак, та має підхід використання каналів стану більш зручним для взаємодій з енергетикою.

1 ОГЛЯД ТА АНАЛІЗ БЛОКЧЕЙН МЕРЕЖ ТА СМАРТ-КОНТРАКТІВ

В цьому розділі буде розібрано структура блокчейна, як він працює. Що таке смарт-контракти та віртуальна машина ефіріум.

1.1 Блокчейн

Спочатку, задля розуміння як працює технологія блокчейн потрібно розповісти як працює p2p мережа. P2P мережа - це по суті мережа рівноправних вузлів, тобто всі учасники цієї мережі рівноправні та виконують функції одразу як і серверів мережі, тобто ті хто обробляє інформацію, як і клієнтів мережі, ті хто робить запити на обробку інформації. Ця архітектура дозволяє створювати розподілені мережі де немає когось одного головного в мережі тобто дозволяє блокчейну бути децентралізованою.

Сатоші Накамото опублікував у 2008 році технічну документацію по біткойну[3], з того часу технологія блокчейн стала найбільш обговорювана у сучасному світу. Блокчейн по суті являється розподіленою базою даних в якій зберігаються всі транзакції системи. Ця база даних формується у вигляді блоків, де кожний створений блок містить хеш попереднього і таким чином створю ланцюг блоків. Все починається з первинного блоку в системі. Створення наступних блоків містить за собою багато інших технологій. Відомо що блок містить відомості про транзакції, дерево їх хешів та хеш попереднього блоку, але за тим, щоб створити наступний блок стоїть система консенсусу, тобто узгодження всіма учасниками мережі що саме цей блок повинен бути утворений.

Блоки

Блоки у блокчейні складаються з наступних параметрів:

- версія блоку
- хеш попереднього блоку
- корінь дерева меркля транзакцій
- дата і час створення блоку
- нонс та біти
- число транзакцій та їх список

Ці всі параметри створюють заголовок блока. Хеш такого заголовка і є хешом блока.

Корінь дерева меркля транзакцій вираховується наступним чином. Спочатку рахуються хеші всіх транзакцій у блоці, потім хеші від суми хешів і так далі. Це зроблено для того, щоб не можна було підробити якусь транзакцію, бо якщо змінити хоча б одну транзакцію, то корінь дерева меркля буде вже інакшим значенням і хеш блоку також. А також це надає перевагу в тому щоб не зберігати всі хеші транзакцій, а лише корінь дерева меркля і таким чином зменшується пам'ять яку займає блок у блокчейні.

Транзакції

Нехай електроні монети будуть представлені як ланцюг цифрових підписів. Кожний власник, щоб відправити монети до іншого користувача повинен підписати хеш попередньої транзакції та публічний ключ наступного власника таким чином монети відправляються наступному власнику. Таким чином можна перевірити за допомогою підпису чи виконалось все правильно і переглянути ланцюг транзакцій. Але є проблема повторного використання монети. Щоб перевірити чи власник монет не підписав дві транзакції та повторно не використав ті самі монети та при цьому це була децентралізована система потрібно щоб транзакції публічно анонсувались, а також щоб більшість інших

учасників погоджувались що саме та транзакція вірна. Для цього в блоках існують позначки часу. До кожного блока приєднується позначка часу яка містить попередню позначку тим самим також формує ланцюг. Це потрібно щоб довести що транзакція існує в часі. Але це все повинно підкріплюватись механізмом консенсусу.

Також щоб зрозуміти краще як виконуються транзакції потрібно розібрати як саме виконується підпис хешів. Це важливо щоб розуміти наскільки це безпечно виконувати та чи можливо підробити підпис. Розглядатись це буде на прикладі мережі ефіріум де підпис виконується завдяки алгоритму ECDSA (Elliptic Curve Digital Signature Algorithm) цей алгоритм побудований на еліптичній кривій. У біткойні та ефіріумі використовують еліптичну криву secp256k1, яка має наступний вигляд:

$$y^2 = x^3 + 7$$

Ця крива розглядається над полем простих чисел F_p, де p просте число, яке дорівнює $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. Так само для secp256k1 визначена точка на цій еліптичній кривій, яка є точкою генерації на даній кривій G. Цей алгоритм є по суті є алгоритмом симетричного підпису, тому кожний користувач повинен мати приватний ключ, а також публічний ключ. Приватний ключ k генерується випадковим чином і є по суті 256 бітним числом. Публічний ключ K генерується завдяки точці генерації тобто $K = k * G$. Далі буде наведено алгоритм генерації цифрового підпису:

Алгоритм 1.1. Алгоритм генерації цифрового підпису

Дано параметри еліптичної кривої E, приватний ключ d та повідомлення m.

1: Обирається випадкове число $k \in [1, p-1]$.

2: Вираховується координата точки $k * G = (x_1, y_1)$.

3: Вираховується $r = x_1 \mod p$, якщо $r = 0$, то вернутись до першого кроку.

4: Вираховується $e = H(m)$, де H - хеш функція.

5: Вираховується $s = k^{-1}(e + d * r) \mod n$, якщо $s = 0$, то вернутись до першого кроку.

6: Вернути (r, s).

Перевірка підпису виконується наступним чином:

Алгоритм 1.2. Алгоритм перевірки цифрового підпису

Дано параметри еліптичної кривої E, публічний ключ Q, повідомлення m та підпис (r, s).

1: Перевірити чи числа r, s є простими числами, якщо ні, то цифровий підпис не вірний.

2: Вираховується $e = H(m)$, де H - хеш функція.

3: Вираховується $w = s^{-1} \mod p$.

4: Вираховується $u_1 = e * w \mod p$, $u_2 = r * w \mod n$.

5: Вираховуються координата точки $X = (x_2, y_2) = u_1 * G + u_2 * Q$.

6: Якщо $X = (0, 0)$ то цифровий підпис не вірний, інакше вирахувати $v = x_2 \mod p$.

7: Якщо $v = r$ то цифровий підпис вірний, інакше не вірний.

Proof of work

Щоб реалізувати систему розділених позначок часу у вигляді p2p, а також блокчейн потрібно щоб всі учасники дійшли до якогось консенсусу. Задля прикладу буде описано концепт proof of work подібний до того який описав Адам Бек у Hashcash[5]. Proof of work застосовує сканування значення яке було хешоване, наприклад SHA-256, початок цього значення починається з нулів. Середня робота розраховується експоненційно у кількостях нулів і може бути перевірена розрахунком лише одного хеша. У блокчейні може бути багато різних ланцюгів, але головний ланцюг це той що найдовший. Це допомагає з атакою повторного використання. Тому що, щоб змінити один блок у блокчейні зловмиснику потрібно

вирахувати хеш попереднього і всіх наступних за ним, і на один більше. А через те, що через proof of work створення блоків є доволі довгою задачею то це зменшує шанс того що таке відбудеться.

Приватність

Звичайні банки, які мають централізовану систему, часто дають приватність тим, що обмежують для інших доступ щодо транзакцій. Хоча в блокчейні транзакції можуть бачити всі, але все одно можна зберегти приватність зробивши публічні ключі анонімними. Тобто всі зможуть побачити що хтось комусь відправляє монети, але транзакції не прив'язані ні до кого. Це можна досягнути якщо генерувати нові пари ключів з кожною транзакцією.

1.2 Віртуальна машина еферіум

Віртуальна машина Ethereum працює самостійно лише задля безпереривного продовження та незмінності операцій станів цієї машини. Стан машини це по суті середовище де всі акаунти та смарт-контракти живуть. Для кожного блока у ланцюзі, Ethereum має лише один канонічний стан і EVM задає правила для розрахунків нових станів від одного блоку до іншого.

Ethereum має свою криптовалюту яка працює майже так само, як і Bitcoin, але ще має додатковий функціонал, наприклад смарт-контрактів. Замість розподіленої бази даних як у біткойні, Ethereum використовує розподілені стани машини. Ці стани являються великими структурами даних які тримають у собі не тільки акаунти і баланси, а ще і машинний стан, який може змінюватись з кожним новим блоком відносно раніше зазначених правилах і також цей машинний стан може виконувати довільний машинний код. Також потрібно сказати що стан є модифікованим деревом Меркля, який дозволяє зв'язувати всі акаунти та

воно зводиться до одного кореневого хешу, що зберігається у блокчейні.

Ethereum має два типи транзакцій: один є механізмом який дозволяє взаємодіяти з іншими контрактами, викликати їх функції, або відправляти криптовалюту іншим акаунтам, інший вид транзакцій дозволяє створити власний смарт-контракт. В результаті створення смарт-контракту створюється новий акаунт у мережі який містить скомпільований байт-код і інші акаунти можуть відправляти транзакції до цього нового акаунта, щоб виконати байт-код.

EVM виконується як стак машина з глибиною 1024 елементів. Кожний елемент це 256 бітне слово, так було обрано задля легшого використання 256 бітної криптографії. Під час виконання EVM зберігає тимчасову пам'ять, яка не передається поміж транзакцій. Смарт-контракти своєю чергою мають пам'ять яка зберігається у вигляді дерева Меркля, які прив'язані до акаунта, а також є частиною глобального стану. Байт-код скомпільованих смарт-контрактів виконується як номери кодів операцій EVM. Коди операцій це правила які зазначені у EVM.

1.3 Смарт-контракт

Смарт-контракт є програмою яка виконується на блокчейні Ethereum. Це набір функцій і даних(станів) які зберігаються на конкретній адресі у блокчейні. Смарт-контракт є акаунтом у блокчейні, тобто це значить що він може мати баланс, а також до нього можна відправляти транзакції. Але цей акаунт не контролюється користувачем, натомість він розгортається у мережі та працює як запрограмовано. Користувач може відправляти транзакції і таким чином викликати функції смарт-контракта. Смарт-контракт можуть визначати правила по якому виконуються транзакції і автоматично забезпечувати їх виконання через код. Смарт-контракт не можна видалити і взаємодіє з ним є незворотнім процесом. Всі смарт-контракти які розгорнуті в Ethereum

можуть бути доступними любому користувачу, навіть більше такі смарт-контракти можуть бути доступні також і іншим смарт-контрактам що дозволяє взаємодіяти одному смарт-контракту з іншим.

Також існують обмеження у смарт-контрактах. Смарт-контракт не може взяти інформацію з якогось зовнішнього ресурсу який не зв'язаний з блокчейном. Покладання на зовнішню інформацію може поставити під загрозу консенсус, який важливий для безпеки та децентралізації. Але за допомогою оракулів це все-таки можливо. Інше обмеження це обмеження в пам'яті, смарт-контракт максимум може мати 24 кілобайт розміру.

Також існують смарт-контракти з множинними підписами - це такі смарт-контракти які вимагають кількох дійсних підписів задля виконання транзакції. Це ефективно якщо потрібно запобігти халепи якщо у смарт-контракті зберігається якась валюта. Також це розподіляє відповідальність за виконання контракту та керування ключами між кількома учасниками запобігає втраті коштів.

Дані у смарт-контрактів зберігаються двома типами. Перший це постійні дані які зберігаються у стані. Тобто ці дані перманентно зберігаються у блокчейні. Другий тип це дані які мають час зберігання виклику функції. Зміна постійних даних буде коштувати додаткових витрат.

Також існують два типи функцій які можуть бути у смарт-контракті, зовнішні та внутрішні. Внутрішні не створюють виклик до EVM і можуть бути доступними лише для виклику лише зі смарт-контракту. Зовнішні абсолютно навпаки, тобто створюють виклик до EVM, і будь-який акаунт у блокчейні може їх викликати. Також вони можуть бути або приватними або публічними. Приватні функції видимі лише для контракту, у якому вони визначені. Публічні навпаки можуть бути викликані зовнішньо. Також кожний смарт-контракт має функцію конструктор, яка виконується лише один раз під час ініціалізації смарт-контракту.

Також в EVM є 4 спеціальних операційних кодів, які дозволяють

взаємодіяти зі смарт-контрактами, 3 з них можуть бути використанні, а саме: call, delegateCall та staticCall. Call - це простий виклик функції. DelegateCall - це виклик функції одного смарт-контракта з другого смарт-контракта з використанням даних другого. StaticCall - це виклик функції в іншому контракті із забороненими змінами стану, такими як створення контракту, випуск подій, модифікація зберігання та знищення контракту.

Для взаємодії програмам які виконуються поза блокчейном смарт-контракт має функціонал подій. Подія може викликатись у функції з параметрами і вони записуються як логи у блокчейн, і ці логи можуть читати програми поза блокчейном.

1.4 Безпека смарт-контрактів

Смарт-контракти — це програми, які знаходяться в децентралізованих блокчейнах і виконуються відповідно до інструкцій, що запускаються. Смарт-контракт діє подібно до традиційної угоди, але заперечує необхідність залучення третьої сторони. Смарт-контракти здатні ініціювати свої команди автоматично, таким чином усуваючи участь органу, що регулює. Як наслідок незмінної функції блокчейна смарт-контракти розробляються у спосіб, який відрізняється від традиційного програмного забезпечення. Після розгортання в блокчейні смарт-контракт не можна змінювати або оновлювати для виправлень безпеки, таким чином заохочуючи розробників впроваджувати надійні стратегії безпеки перед розгортанням, щоб уникнути потенційного використання надалі. Однак останні жахливі атаки та численні наявні вразливості, які є наслідком відсутності латків безпеки, поставили під сумнів сталість цієї технології. Такі атаки, як атака на децентралізовану автономну організацію (DAO) і злом гаманця, коштують мільйони доларів просто через наївні помилки в коді смарт-контракту. У цьому підрозділі розглядаються методи використання блокчейну на основі

наступних атак: атаки на протоколи консенсусу, атака на помилки в смарт-контракті, зловмисне програмне забезпечення, запущене в операційній системі, і атаки створені шахрайськими користувачі. Потім буде розібрано вразливості смарт-контрактів прикладом з 7 найважливіших методів атак.

Зловмисне програмне забезпечення

Зловмисні дії можуть включати розповсюдження зловмисного програмного забезпечення з метою обману користувачів. Цей тип експлуатації здебільшого ініціюється через Інтернет, щоб скомпрометувати особистість користувача або здійснити шахрайство за допомогою шкідливих програм або вірусів. Такі зловмисні дії можуть серйозно вплинути на фінансовий стан жертви [30]. Зловмисні атаки можуть виникати в будь-якій формі, як-от електронний лист із гаманця з проханням синхронізувати обліковий запис із мережею, яка щойно була розгалужена. Експлуатація гаманців користувачів шляхом зловмисних атак може дозволити зловмиснику витягнути всю валюту. Крипто-джекінг, атаки на слабину та форуми – це кілька шкідливих методів, які просять майнерів входити через пошкоджені посилання [6]. Glupteba — ще одне зловмисне програмне забезпечення, яке використовує блокчейн біткойн для свого оновлення. Таким чином, він залишається активним, попри те, що антивірус розриває підключення до сервера. Це зловмисне програмне забезпечення поширюється через сценарії для викрадення конфіденційної інформації, такої як ідентифікатор користувача, паролі, історія веб-перегляду, збережені файли кукі[7].

Атаки створені шахрайськими користувачі

Протокол PoW передбачає, що 50% мережевих майнерів завжди будуть чесними майнерами. Таким чином, зловмисники, які містять більше ніж

50% хешування, можуть отримати контроль над мережею [8]. Слабкий консенсус також може призвести до численних атак, пов'язаних з мережею блокчейн. Атака Sybil дозволяє зловмиснику встановити кілька шкідливих вузлів у мережі блокчейну Bitcoin. Потім шкідливі вузли використовуються для пошкодження мережі, проведення непривілейованих транзакцій або зміни дійсних транзакцій. Подібним чином, атака Eclipse може бути виконана для маніпулювання одноранговою мережею (P2P), щоб отримати повний контроль над інформацією, яку містить вузол. Крім того, викрадення протоколу прикордонного шлюзу (BGP) робить помилкові декларації в системі маршрутизації, щоб перенаправити трафік. Таким чином, незважаючи на децентралізовану функцію, мережа блокчейну все ще може бути скомпрометована різними методами атак через слабкий консенсус.

Атаки створені шахрайськими користувачі

Ця експлуатація обманює продавців, щоб скористатися нестабільністю цифрових транзакцій. Шахрайство може змусити продавця випустити товари до повного підтвердження транзакції. У звичайному сценарії транзакція Bitcoin підтверджується після 6 транзакцій. Однак споживач може переконати торговця відпустити товар, не чекаючи до 6 транзакцій, щоб можна було застосувати такі методи атаки, як 1 підтвердження або n підтвердження, щоб подвоїти витрати. Подібним чином, останнім часом різні роздрібні продавці приймають криптовалюти, що дозволяє споживачам миттєво отримувати їхній продукт [9]. Наприклад, придбати каву в кав'ярні. Розглянемо сценарій, коли зловмиснику вдається витратити ту саму криптовалюту за короткий проміжок часу, що почне змагання між обома транзакціями. Якщо друга транзакція буде прийнята майнерами пулу для обробки, то першу транзакцію буде відхилено, потенційно залишивши торговця без оплати за надані товари.

Атака на помилки в смарт-контракті

Експлуатація помилок програми виникає, коли в коді смарт-контракту є помилка. Ця експлуатація в основному відбувається в смарт-контрактах. Це виникає, коли розробники не можуть визначити помилки коду в децентралізованому додатку. Зловмисники можуть витягнути всі гроші з гаманця контракту через прості помилки коду. Програми смарт-контрактів схожі на веб-програми, які працюють через блокчейн. Як і веб-програми, вони також можуть містити помилки, однак ці помилки можуть призвести до серйозних проблем. Наприклад, DAO зміг зібрати 150 мільйонів доларів, тоді як зловмисник зміг вкрасти близько 60 мільйонів доларів через помилки в коді. [10]. Rubixi та GovernMental є одними з програм смарт-контрактів, які мали недоліки через помилки коду [11]. Помилки програми можуть не тільки дозволити зловмисникам викрасти гроші, але й вплинути на те, щоб програма працювала по-іншому.

1.5 Методи атак на смарт-контракти

У цьому розділі розглядається сім методів атаки, які можуть мати серйозний вплив на програму смарт-контракту. Успішне виконання таких атак може призвести до неочікуваної роботи смарт-контракту. Отже, сторони, пов'язані з контрактною угодою, можуть зазнати серйозних збитків.

Повторне використання функції

Повторне використання функції вважається однією з найбільш катастрофічних технік атак у смарт-контрактах [12]. Ця техніка атаки здатна повністю зруйнувати договір або викрасти цінну інформацію. Цей метод атаки може виникнути, коли функція викликає інший контракт

через зовнішній виклик. Експлуатація дозволяє зловмиснику виконати рекурсивний зворотний виклик основної функції, створюючи ненавмисний цикл, який повторюється багато разів. Наприклад, коли вразливий контракт містить функцію відкликання, контракт може незаконно викликати функцію відкликання багато разів, щоб витратити будь-який доступний баланс, який містить контракт. Атаки повторного використання з однією функцією та атаки повторного використання між функціями — це два різні типи, якими можуть скористатися зловмисники. Експлуатація дозволяє зловмиснику використовувати зовнішні виклики для виконання бажаних завдань.

Переповнення параметрів смарт-контракту

Цю вразливість відносно легко ініціювати, і вона виникає в транзакціях, які приймають неавторизовані вхідні дані або значення [13]. Переповнення смарт-контракту в основному відбувається, коли надається більше значення, ніж максимальне значення [14]. Контракти в основному написані в Solidity, яка може обробляти до 256-бітних чисел, тому збільшення на 1 призведе до переповнення. Традиційні підходи до тестування недостатні для визначення вразливості до цього методу атак.

Атака на коротку адресу

Ця вразливість виникає через слабкість віртуальної машини Ethereum (EVM) [15]. EVM дозволяє використовувати неточні доповнені аргументи, що дозволяє зловмисникам надсилати спеціально створені адреси, які призводять до експлуатації. Атака на коротку адресу використовує подібну стратегію атаки, як помилка впровадження SQL [16]. Коли виявляється переповнення, EVM включає нуль у кінці адреси, щоб переконатися, що вона містить 256-бітний тип даних. Однак зловмисник може скористатися цією вразливістю, опустивши останній нуль в адресі

ether. Ця вразливість є помилкою перевірки вхідних даних і в основному виникає з боку відправника через слабкий код генерації транзакцій.

DelegateCall

Розробники смарт-контрактів використовують CALL і DELEGATE-CALL для модульного написання коду [17]. Код операції DELEGATE містить функцію, подібну до повідомлення CALL, однак, окрім коду, який виконується для виклику контракту, msg.sender і msg.value не змінюються. Цей атрибут дозволяє розробникам генерувати багаторазовий код, підвищуючи ймовірність раптового виконання коду за допомогою DELEGATECALL. Функція DELEGATECALL показує, що під час створення користувальницьких бібліотек можна створити недоліки, а також це може призвести до нових вразливостей. Уразливості DELEGATECALL можна уникнути, спостерігаючи за порушеннями як у контракті про бібліотеку, так і в контракті про виклик, і, крім того, розробляючи бібліотеки без стану, коли це можливо.

Атаки на специфікатори видимості

Специфікатори видимості у функції Solidity контролюють спосіб виклику функції [17]. Специфікатор видимості також бере на себе контроль, коли дозволяє користувачам викликати зовнішні функції за допомогою похідних контрактів. Неправильна реалізація специфікаторів видимості може спричинити серйозні наслідки для смарт-контракту. Видимість за замовчуванням завжди встановлена, як публічна для функцій, що дозволяє зовнішнім контрактам вимагати видимості, якщо функції не згадують про це явно. Ця вразливість виникає, коли розробники нехтують установкою специфікатора видимості на приватний.

Атаки послідовними транзакціями

Залежність від послідовності транзакцій — це вразливість, яка може дозволити корумпованим майнерам мати серйозний вплив на смарт-контракти [18]. Ця вразливість є дуже поширеною помилкою безпеки в смарт-контракті, що залежить від порядку виконання транзакцій [19]. Наприклад, щойно згенерований блок містить 2 транзакції, що забезпечують виконання одного смарт-контракту. Такі графіки не надають користувачам достатньо інформації, щоб визначити стан контракту або коли ініціюється індивідуальний виклик. Тому, коли результат обох транзакцій залежить від стану, контракт призводить до цієї уразливості.

У блокчейні Ethereum майнери відповідають за контроль порядку транзакцій, віддаючи пріоритет транзакціям з більшим газом. Отже, будь-який майнер, який закриває блок, може вплинути на порядок транзакції. Здатність потенційних майнерів впливати на порядок транзакцій для незаконної діяльності є результатом залежності послідовності транзакцій.

Залежність від дати і часу створення блоку

Залежність від часової позначки — ще одна вразливість, якою можуть скористатися корумповані майнери [18]. Щоб отримати вигоду, майнер може змінити мітку часу на кілька секунд. Уразливість залежності часової позначки виникає через неправильне розуміння хронометражу [46]. Це дозволяє від'єднати мережу Ethereum від синхронізованого глобального годинника. Наприклад, смарт-контракт використовує поточну позначку часу для створення випадкових чисел для визначення результату лотереї. Оскільки смарт-контракт дозволяє майнерам розміщувати позначку часу протягом 30 секунд після перевірки блоку, це дає майнерам більше можливостей для експлуатації. Отже, результат генератора випадкових чисел можна змінити, щоб отримати переваги.

Висновки до розділу 1

У цьому розділі було розібрано як працює блокчейн, EVM та смарт-контракти. Це допомагає зрозуміти базову структуру, задля наступних розділів, де потрібно напряму працювати з EVM, та реалізовувати смарт-контракт. Також було розглянуто методи атаки на смарт-контракт, щоб запобігти можливостям використання таких методів у реалізованому смарт-контракті. І з цього розділу можна було дізнатись, що атаки на смарт-контракти є доволі різними і частими у мережі блокчейн, тому це дуже клопітка робота реалізувати це правильно, без можливостей взлому. Також потрібно звернути увагу на те що, пам'ять блокчейна на зберігання смарт-контракта є обмеженою, тому потрібно використовувати строго типізовані дані у смарт-контрактах.

2 ВИКОРИСТАННЯ СМАРТ-КОНТРАКТІВ В ЕЛЕКТРОЕНЕРГЕТИЦІ

У цьому розділі досліджується використання блокчейн технологій в електроенергетиці, а також способи реалізації смарт-контрактів у електроенергетиці, та наявні приклади.

2.1 Використання блокчейн технологій в електроенергетиці

Енергетичні ринки стикаються зі змінами, спричиненими технологічним і соціально-економічним розвитком. Створення електроенергетики переходи від звичайних теплових електростанцій до розподілених енергетичних ресурсів. Це викликає коливання пропозиції, посилюючи невизначеність. Торгівля електроенергетикою стає складнішою. Відповідно пропозиція і ціна схильні до високого рівня невизначеності. Розподілені енергетичні ресурси, наприклад електростанції, фотоелектричні установки і біогазові установки стають все більш популярними серед звичайних людей та місцевих організацій. Потреба в електроенергії може бути забезпечена локально, створюючи електромережу реалізацією складного рівня. Зміна енергетичного ринку збільшує участь на ньому для клієнтів, а також збільшує можливості використання, що дозволяє клієнтам оптимізувати споживання та збільшити кількість продавців електроенергії [21].

2.2 Канали стану

Першими протоколами поза мережею були платіжні канали Bitcoin, завдяки Спілману [22]. У платіжному каналі Аліса відкриває канал для Боба, ініціювавши депозитну транзакцію в мережі, прив'язуючи суми

депозиту до програми смарт-контракту. Потім дві сторони можуть здійснювати між собою довільну кількість швидких платежів, просто обмінюючись підписаними повідомленнями поза мережею. Для закриття потрібна остання транзакція в мережі каналу, який потім розподіляє остаточний баланс відповідно до коду смарт-контракта платіжного каналу.

Платіжні канали Спілмана дозволяють лише однонаправлені платежі (тобто ролі відправника та одержувача мають бути фіксованими при створенні каналу). Далі канали стану розробляли Декер і Ваттенхофер [23], а також Пун і Дрія [24] які реалізували «дуплексні» платежі від будь-якої сторони до іншого. Через обмеження сценарію Bitcoin ці конструкції і вимагали складних обхідних шляхів (наприклад, для каналів Пуна і Дрія потрібні були сторони для зберігання постійного списку, який збільшується, ключів відкликання для захисту від зловмисників). Простіша конструкція платіжних каналів була розроблена для мережі Ethereum на основі підписів круглих чисел [25]. Для простоти якраз розглянуто лише останній підхід. Протокол платіжного каналу поза мережею складається з трьох наступних фаз:

Відкриття каналу. Канал спочатку відкривається транзакцією депозиту у мережі. Це резервує певну кількість цифрової валюти та прив'язує її до програми смарт-контракта.

Платежі поза мережею. Щоб здійснити платіж поза мережею, сторони обмінюються підписаними повідомленнями, що відображають оновлений баланс. Наприклад, поточний стан буде представлено як підписане повідомлення $(\sigma A, \sigma B, i, A, B)$, де пара підписів σA і σB дійсні для повідомлення (i, A, B), де A (відповідно B) — баланс Аліси (відповідно Боба) у раунді i. Кожна сторона локально відстежує поточний баланс, що відповідає останньому підписаному повідомленню.

Розгляд суперечок. Смарт-контракт блокчейну (до якого прив'язана депозитна транзакція) служить «розпорядником суперечок». Це відбувається, коли будь-яка сторона підозрює невдачу або бажає

закрити канал і зняти залишок. Обробник диспутів залишається активним протягом фіксованого часу, протягом якого будь-яка сторона може надати докази (наприклад, підписані повідомлення). Розпорядник суперечок приймає докази і виплачує гроші відповідно.

2.3 Мережа Рейден

У цьому підрозділі описується мережа блокчейн в якій вже реалізовано канали стану як функціонал мережі. Тому надалі буде розібрано як саме це було реалізовано в цій мережі.

Посередницькі перекази

Посередницькі перекази використовують кілька платіжних каналів для завершення платежу, що дозволяє користувачам здійснювати платежі з іншими користувачами, з якими вони не обов'язково мають платіжний канал.

Як це працює буде розібрано на прикладі: Аліса надсилає платіж Дейву, використовуючи канали між нею, Бобом, Чарлі та Дейвом. Цей платіж заблоковано як незавершений переказ, що означає, що для нього потрібен секрет, перш ніж його можна буде вимагати для отримання токенів. Отримавши передачу, Дейв запитує секрет у Аліси. Аліса надсилає Дейву секрет, який він використовує, щоб розблокувати незавершену передачу. Дейв надсилає секрет Чарлі, який підписує підтвердження балансу, що відображає додаткову вартість розблокованого незавершеного переказу, і надсилає його Дейву. Потім протокол виконується у зворотному напрямку, поки секрет не буде розкрито всім на маршруті посередницької передачі.

Якщо один із вузлів-посередників перестає відповідати (наприклад, через те, що він перебуває в автономному режимі) після запиту секрету, неможливо буде розблокувати всі незавершені передачі до закінчення

терміну дії блокувань.

У такому випадку Дейв може натомість подати секрет до смарт-контракту під назвою секретний реєстр. Після реєстрації секрету в ланцюжку вузли-посередники можуть побачити секрет, розблокувати свої перекази та надіслати підписане оновлене підтвердження балансу своєму контрагенту.

Маршрутизація

Для успішного здійснення посередницького переказу між відправником і одержувачем потрібен шлях платіжних каналів із достатньою пропускною здатністю.

Пропускна здатність означає, що платіжні канали, які використовуються для платежу, мають достатню кількість токенів для створення незавершених переказів у всіх необхідних каналах.

У прикладі для цього випадку Аліса хоче заплатити Дейву 3 токени, тому їй потрібно знайти маршрут підключених каналів із пропускною здатністю 3 у посередницьких платіжних каналах.

Протокол мережі Рейден має повну картину мережі та всі початкові баланси каналів. Припустимо, що всі користувачі внесли по 5 токенів на канал і вже зробили кілька переказів. Потім протокол пробує різні шляхи, поки не буде знайдено той із достатньою місткістю.

Обробка кількох незавершених переказів

Щоб керувати декількома незавершеними передачами, вузли зберігають локальний стан у структурі дерева Меркля. У мережі Рейден кожен вузол має одне дерево Меркля на канал і напрямок.

Під час опосередкованої передачі кожен вузол-посередник додає опосередковану передачу до свого дерева Меркля. Коли створюються нові незавершені передачі, вузол-посередник перевіряє, чи корінь дерева

відповідає їхньому локальному стану. Така конструкція дозволяє протоколу обробляти 160 незавершених передач на канал. Обмеження до 160 — це округлене значення, яке гарантує, що вартість газу для розблокування становитиме менше ніж 40% від традиційного ліміту блокового газу Ефіріуму в мільйон пі.

Розблокування кількох незавершених переказів поза мережею

Вузли можуть перестати відповідати під час незавершеної передачі, що призводить до того, що секрет реєструється в ланцюжку в секретному реєстрі. Для цього в цій мережі є протокол який з цим розбирається.

Процес розрахунку незавершених блокувань гарантує, що попередньо виділені кошти, які були захищені блокуваннями, розблоковуються або для одержувача, або для відправника платежу. Напрямок визначається тим, чи відома таємниця. Якщо секрет розкривається поза ланцюгом, то взаємодія в ланцюзі не потрібна. Якщо контрагент не відповідає, блокчейн завжди можна використовувати, щоб гарантувати безпечні перекази.

Висновки до розділу 2

У цьому розділі було розглянуто канали стану, а саме сама їх структура, що допомагає у розумінні як саме це можна реалізувати у власному смарт-контракті і надати безпечну взаємодію між продавцем і покупцем. Також було розібрано те, як може працювати блокчейн у ринку електроенергетики, тобто надати їй децентралізованості і можливості користувачам безпечно та ефективно використовувати електроенергетику. І як основа, було розібрано реалізацію каналів стану у мережі Рейден, як логічно цей процес побудований, з нюансами взаємодії продавців та покупців і запобіганню шахрайства. Також через мережу Рейден і як працює блокчейн, можна побачити, що цікавою задачею є як обрати кількість токенів для заморожування, щоб це було і не занадто

мало, щоб користувачі не шахраювали і не виконували свої обов'язки по покупці, і також не занадто багато, щоб користувачам не було думки що це того не варто. Ця вся інформація дозволяє розробити власний смарт-контракт, під продаж електроенергетики.

3 РЕАЛІЗАЦІЯ СМАРТ-КОНТРАКТУ ДЛЯ ВЗАЄМОДІЇ З ЕЛЕКТРОЕНЕРГЕТИКОЮ

В цьому розділі буде описано. як повинний смарт-контракт працювати в електроенергетиці за допомогою каналів, структуру смарт-контракта, опис функцій, взаємодію з продавцем та покупцем.

3.1 Бізнес-логіка взаємодії продавця електроенергетики та покупця

Для початку, щоб реалізувати смарт-контракт, потрібно зрозуміти як він повинен працювати. З однієї сторони у нас є продавець електроенергетики, з іншим покупцем. Один повинен продати електроенергетику інший купити. Система яка запропонована нижче, грає роль третьої особи, що дозволяє своєю чергою бути валідатором процесу, а через те, що ця третя особа це смарт-контракт то валідатором є усі учасники блокчейну.

Для взаємодії по валюті і передачі використовуються ERC20 токени, це зроблено задля того, щоб саме смарт-контракт міг ними управляти. ERC20 токени це також по суті смарт-контракт, який має наступний функціонал по використанню:

- transfer - відправляє токени з адреси що викликає функцію на інший адрес

- transferFrom - відправляє токени з одного адресу на інший(якщо це дозволено адресом з якого відправляють токени)

- approve - дозволяє іншому адресу використовувати токени того хто викликав функцію, кількість залежить від зазначених параметрів

- allowance - вертає скільки токенів дозволено використовувати адресу що витрачає чужі кошти

- increaseAllowance - збільшує кількість токенів, які дозволені на витрату

- decreaseAllowance - зменшує кількість токенів, які дозволені на витрату

Таким чином у продавця та покупця повинна бути цей токен, задля того, щоб можна було обмінятись валютою. Також у покупця повинен стояти контролер, який буде підключений до електромережі, а також до бекенду мережі, який своєю чергою буде робити виклики до блокчейну, щоб відправляти скільки електроенергетики було витрачено. Задля безпеки смарт-контракт повинен використовувати також канали стану, це дозволяє прописати договір між продавцем та покупцем, щоб ніхто з них не мав змогу отримати або токени, або послуги і при цьому не надати своїх. Канал стану у цьому випадку повинен працювати наступним чином, покупець має бажання купити електроенергетику у продавця, він з ним підписує контракт, під час якого покупець надсилає свої кошти(токени) у смарт-контракт відносно того скільки він хоче отримати електроенергії. Після цього продавець надає електроенергію протягом якогось терміну, який зазначений у смарт-контракті, як термін каналу, після того, як термін пройшов, покупець або продавець може закрити канал під час цього йде виклик до контролера і він відправляє скільки електроенергії було витрачено, зі сторони покупця, до смарт-контракту, смарт-контракт своєю чергою повинен порахувати скільки було витрачено електроенергії та перерахувати кошти, з тих що були занесені покупцем, продавцю, а решту вертає покупцю.

Новий підхід використання каналів стану полягає, у тому щоб використовувати заморожені смарт-контрактом кошти задля розплати за електроенергію, це дозволяє більш адаптуватись саме до використання каналів стану у електромережі, а також це дозволяє усунути проблему депозита коштів які повинні бути заморожені, а саме їх кількістю, і працювати лише з тими коштами які по суті вже є оплатою за електроенергію.

Така система дозволяє позбутись шахрайств у взаємодії продавця і покупця. Наприклад у випадку, коли продавець зовсім не надав електроенергетики то контролер верне 0 як використану електроенергію смарт-контракту і таким чином покупцю не прийдеться нічого платити. Або не може бути випадків, щоб покупець не заплатив за використану електроенергетику, через те, що контролер викликає функцію вертання токенів, коли термін каналу закінчується і продавець по суті закинув вже свої токени смарт-контракту.

3.2 Реалізація смарт-контракту

Смарт-контракт був написаний на мові Solidity, повний код програми можна побачити у додатку А.2. Далі буде розібрано функціонал цього смарт-контракту з деякими поясненнями, для початку потрібно було реалізувати структуру даних PaymentChannel яка своєю чергою має наступні змінні:

- sender - адреса покупця

- senderDeposite - змінна яка перевіряє чи надіслав покупець токени

- receiver - адреса продавця

- tokenValue - кількість токенів відносно електроенергії, як домовленість між покупцем та продавцем, скільки покупець хоче купити електроенергії

- electricityValue - кількість максимальної електроенергії яку може витратити покупець

- settlingPeriod - час скільки канал стану повинен бути відкритим

- settlingUntil - час коли канал закривається

Для зберігання цих структур використовується змінна channels, яка по суті є хеш таблицею, де відносно якогось числа зберігається структура даних PaymentChannel.

По функціях смарт-контракта є наступні:

- open - ця функція відповідає за створення каналу і має наступні

параметри channelId(номер за яким зберігається цей канал), receiver(адрес продавця), settlingPeriod(час скільки канал стану повинен бути відкритим), tokenValue(кількість токенів відносно електроенергії, як домовленість між покупцем та продавцем, скільки покупець хоче купити електроенергії). Для того, щоб ця функція виконувалась, спочатку потрібно викликати у ERC20 токені функцію approve з адресою смарт-контракта. Так, наступним чином ця функція створює канал стану, а також викликає emit про те що канал було створено.

- deposite - ця функція дозволяє перекинути токени від покупця до смарт-контракту і має наступний параметр channelId(номер за яким зберігається канал). У цій функції реалізована перевірка, щоб інша ніяка людина не могла внести токена за покупця, а також змінює senderDeposite на true і викликає emit про те що токени було відправлено.

- startSettling - ця функція активує канал, тобто взаємодію між продавцем і покупцем, і має наступний параметр channelId(номер за яким зберігається канал). У цій функції перевіряється чи функцію викликав покупець, бо саме він може почати цей процес, а також розраховується settlingUntil, тобто кінцева дата, коли канал може бути знищеним.

- canClaim - ця функція має декілька перевірок, а саме на те чи канал можна знищити, чи це викликалось зі сторони бекенду і чи кількість токенів не більша за можливе використання і має наступні параметри: channelId(номер за яким зберігається канал), payment(кількість використаних токенів), origin(адреса яка потім перевіряється чи це викликав контролер), electricitySpend(скільки електроенергії було витрачено), signature(підпис який підписує бекенд, тим самим підтверджує, що це легітимна транзакція).

- Claim - ця функція знищує канал, розраховуючи витрачену електроенергію та кошти які потрібно відправити продавцю, і вернути за невикористану електроенергію покупцю. І має наступні параметри: channelId(номер за яким зберігається канал), payment(кількість використаних токенів), origin(адреса яка потім перевіряється чи це

викликав контролер), electricitySpend(скільки електроенергії було витрачено), signature(підпис який підписує бекенд, тим самим підтверджує, що це легітимна транзакція). А також в ній викликається функція canClaim для перевірки всіх нюансів.

Реалізований смарт-контракт з даним функціоналом задовільняє безпечній взаємодії продажу і покупці електроенергії. Він також дозволяє децентралізувати продаж електроенергетики і заохочує різних учасників брати у цьому участь. Таким чином, може створитись ринок продажі електроенергетики.

Даний смарт-контракт також був перевірений на функціональність на тестовій мережі Ethereum з емуляцією бекенда і роботи контролера.

Також реалізований смарт-контракт та бізнес-процес можна покращити, наприклад можна додати вибір необхідної потужності для електроенергетики.

3.3 Перевірка безпеки смарт-контракту на основні атаки

Далі будуть наведені основні атаки, які були зроблені, на реалізований смарт-контракт, та реакція щодо цього.

Повторне використання функції

Як було вже розібрано, цей вид атаки можна виконати коли функція викликає інший контракт, саме так відбувається у функції claim у реалізованому смарт-контаркті. Для цього було додано бібліотеку ReentrancyGuardUpgradeable, у якому реалізована перевірка на виконання функції, і ця перевірка запобігає повторному використанню цієї функції з самої функції.

Задля перевірки цього було написано простий смарт-контракт А.2, який має fallback функцію яка ще раз викликає ту ж саму функцію. Але оскільки стоїть перевірка на те, щоб це не відбувалось, то цей вид атаки

неможливий.

Переповнення параметрів смарт-контракту

Цей вид атаки неможливо виконати у реалізованому смарт-контракті лише з однієї причини. У смарт-контракта хоч і є параметри які можна переповнити, але зловмисник ніяк не зможе це виконати, оскільки усі функції смарт-контракту приймають параметри лише ззовні і не створюють нові чи не збільшують або зменшують значення всередині функцій.

Атака послідовними транзакціями

У реалізованому контракті важливо щоб транзакції виконувались послідовно, тобто, наприклад буде погано якщо функція claim виконається до того як виконається функція startSettling. Задля запобігання таких атак було додано булеві зміни, які перевіряють чи у правильній послідовності виконуються ті, чи інші функції.

Задля перевірки коректності було це перевірено на тестовій мережі Ethereum, а саме випадки чи може виконуватись startSettling перед deposite, а також claim перед startSettling. В таких випадках контракт кидає помилки.

Залежність від дати і часу створення блоку

Реалізований смарт-контракт залежить від мітки часу наступним чином, це дозволяє тримати канал відкритим певний період і дійсно зловмисник зможе змінити мітку часу на кілька секунд, щоб швидше викликати функцію claim, але це не дасть ніякого результату, оскільки ціна яку повинен заплатити користувач за користуванням електроенергії, повністю залежить від того скільки саме він ту електроенергію витратив.

Висновки до розділу 3

У цьому розділі описувалась бізнес логіка, яку повинен використовувати смарт-контракт, для взаємодії між покупцем та продавцем електроенергетики. Також був реалізований даний смарт-контракт і описаний його функціонал. Реалізований даний смарт-контракт був на основі каналі станів, але з більш зручним підходом для використання в енергетиці, тобто використовував логіку заморожування токенів, допоки процес передачі електроенергії не був виконаний, після цього розподілялись кошти відносно того скільки електроенергії було використано. Також можна дійти до висновку, що даний смарт-контракт може бути поліпшений відносно потрібної реалізації, тобто можна додати регулювання потужностей електроенергії. А також було перевірено смарт-контракт на основні типи атак і як висновок, можна прийти до того, що основні атаки не можуть бути виконані щодо реалізованого смарт-контракту.

ВИСНОВКИ

Як результат оглядової частини було розібрано як працює блокчейн мережа, віртуальна машина ефіріум та смарт-контракти. Ця інформація допомогла зрозуміти структуру на чому все побудовано, а також при реалізації власного смарт-контракту. Ще було розглянуто методи атак на смарт-контракт, і які в цілому бувають атаки. Знання цієї інформації дозволяє реалізовувати смарт-контракти так, щоб їх не можна було використовувати задля заволодіння чужими токенами, чи можливо навіть підлаштування смарт-контракта під себе. Тому проаналізувавши цю інформацію можна реалізувати якомога безпечний смарт-контракт.

У другому розділі було розібрано як працюють канали стану. Це було зроблено задля того, щоб реалізувати у смарт-контракті такий канал стану, який безпечно дозволяє взаємодіяти учасникам при торгівлі електроенергії. Також було досліджено як працює ця безпека. В цілому вона заморожує кошти учасників, допоки взаємодія між ними не буде закінченою. Ще було розібрано приклад реалізацій канал стану на мережі Raiden. Це допомогло зрозуміти, як практично такий канал стану працює і розібратись те що канали стану, крім цього також є доволі новими у блокчейні, тому що, ще багато є нюансів яких можна там покращити. Наприклад вибір правильного депозиту коштів учасників, це складне питання, тому що відносно цього залежить, наскільки безпечно і комфортно користуватись тим чи іншим каналом стану.

Як результат, виконаної вище роботи, було реалізовано смарт-контракт який використовує свій підхід до каналів стану, задля того, щоб безпечно та дешевше використовувати смарт-контракти в енергетиці. Так, як смарт-контракт використовує заморожені кошти у каналі стану, не просто задля того, щоб дотримувались умови взаємодії між продавцем і покупцем, а також щоб покупець міг оплатити цими коштами за електроенергію, це допомагає розв'язати проблему вибору

депозиту, оскільки він у цьому випадку просто буде залежати від максимальної кількості електроенергії яку хоче витратити покупець. Також це дозволяє зробити продажу електроенергетики децентралізованою, тобто будь-який користувач, який має якийсь ресурс створення електроенергії, зможе його продавати за допомогою цього смарт-контракту. А також як проєкт надалі, цей смарт-контракт може бути покращений тим, що можна додати функціонал, який би контролював різні аспекти електроенергетики, наприклад напруги. Зараз є невелика проблема з тим, яка саме потужність подається тим чи іншим приладам, якщо в один момент у мережі йде великий виріст напруги, то це може знищити деякі електроприлади, але за допомогою цього функціоналу можна буде перевіряти яка напруга подається і можливо навіть якось її корегувати.

ПЕРЕЛІК ПОСИЛАНЬ

1. Державний земельний кадастр перейшов на технологію Blockchain [Електроний ресурс] / Режим доступу: https://web.archive.org/web/20171003174819/http://minagro.gov.ua/node/24722

2. Journal of Cryptology 3 [Електроний ресурс] / Haber, Stuart, Stornetta, W. Scott — 2008. — Режим доступу: https://link.springer.com/article/10.1007/BF00196791

3. Bitcoin whitepaper. [Електроний ресурс] / Satoshi Nakamoto. — 2008. — Режим доступу: https://bitcoin.org/bitcoin

4. Smart contracts in the Netherlands [Електроний ресурс] / Schulpen, Ruben R.W.H.G. — 2018. — Режим доступу: http://arno.uvt.nl/show.cgi?fid=146860

5. Hashcash - a denial of service counter-measure [Електроний ресурс] / A. Back. — 2002. — Режим доступу: http://www.hashcash.org/papers/hashcash.pdf

6. J. J. Xu *Are blockchains immune to all malicious attacks?*. 2016.

7. Warning Issued After Malware Is Found To Have Hijacked Bitcoin Blockchain [Електроний ресурс] / B. Bambrough. — 2019. — Режим доступу: https://www.forbes.com/sites/billybambrough/2019/09/07/serious-malware-warning-over-bitcoin-blockchain/#cc2d8347c286.

8. S. Sayeed and H. Marco-Gisbert *Assessing blockchain consensus and security mechanisms against the 51% attack*. 2019.

9. Customers Can Spend Bitcoin At Star- bucks, Nordstrom And Whole Foods [Електроний ресурс] / M. del Castillo. — 2019. — Режим доступу: https://www.forbes.com/sites/michaeldelcastillo/2019/05/13/starbucks-nordstrom-and-whole-foods-now-accept-bitcoin-just-dont-ask-them/659a4e592252.

10. The DAO Hack Explained: Unfortunate Take-off of Smart Contracts [Електроний ресурс] / O. G. GÃijçlütürk.

— 2018. — Режим доступу: https://medium.com/@ogucluturk/the-dao-hack-explained-unfortunate-take-off-of-smart-contracts-2bd8c8db3562.

11. N. Atzei, M. Bartoletti, and T. Cimoli *A survey of attacks on ethereum smart contracts.*

12. Protect Your Solidity Smart Contracts From Reentrancy Attacks [Електроний ресурс] / W. Shahda. — 2019. — Режим доступу: https://medium.com/coinmonks/protect-your-solidity-smart-contracts-from-reentrancy-attacks-9972c3af7c21.

13. J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen *Easyflow: Keep ethereum away from overflow*. 2019.

14. S. Sayeed and H. Marco-Gisbert *On the effectiveness of control-ow integrity against modern attack techniques*. 2019.

15. Blockchain Attack Vectors: Vulnerabilities of the Most Secure Technology [Електроний ресурс] / A. Bryk. — 2018. — Режим доступу: https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors.

16. ICO Smart contract Vulnerability: Short Address Attack [Електроний ресурс] / S.Esra. — 2018. — Режим доступу: https://medium.com/huzzle/ico-smart-contract-vulnerability-short-address-attack-31ac9177eb6b.

17. Solidity Security: Comprehensive list of known attack vectors and common anti-patterns [Електроний ресурс] / A. Manning. — 2018. — Режим доступу: https://blog.sigmaprime.io/solidity-security.html.

18. Smart contract security issues: what are smart contract vulnerabilities and how to protect [Електроний ресурс] / S. Pro. — 2018. — Режим доступу: https://smartym.pro/blog/smart-contract-security-issues-smart-contract-vulnerabilities-and-how-to-protect/.

19. A.Das,S.Balzer,J.Hoffmann,andF.Pfenning *Resource-awaresession types for digital contracts*. 2019.

20. Why Smart Contracts Fail: Undiscovered bugs and

what we can do about them [Електроний ресурс] / H. Olickel. — 2016. — Режим доступу: https://medium.com/@hrishiolickel/why-smart-contracts-fail-undiscovered-bugs-and-what-we-can-do-about-them.

21. Dalkmann, U *In Smart Market - Vom Smart Gridzum intelligenten Energiemarkt*. 2014.

22. Anti dos for tx replacement. [Електроний ресурс] / Jeremy Spilman. — 2013. — Режим доступу: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html

23. Christian Decker and Roger Wattenhofer *A fast and scalable payment network with bitcoin duplex micropayment channels*. 2015.

24. The bitcoin lightning network: Scalable off-chain instant payments [Електроний ресурс] / J. Poon and T. Dryja. — 2013. — Режим доступу: https://lightning.network/lightning-networkpaper.pdf.

25. Instantaneous decentralized poker [Електроний ресурс] / Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. — 2017. — Режим доступу: https://arxiv.org/abs/1701.06726,2017.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 Реалізований смарт-контракт для взаємодії з електроенергетикою

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts-
upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-
upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-
upgradeable/security/ReentrancyGuardUpgradeable.sol";


contract Controller is Initializable, AccessControlUpgradeable,
ReentrancyGuardUpgradeable {

    address private _owner;

    struct PaymentChannel {
        address sender;
        bool senderDeposite;
        address receiver;
        uint256 tokenValue;
        uint256 electricityValue;

        uint256 settlingPeriod;
        uint256 settlingUntil;
    }
```

```
mapping (bytes32 => PaymentChannel) public channels;

IERC20Upgradeable public token;

uint256 private depositeValue;

event Convert(address account, uint256 tokenId);
event OpenChannel(
bytes32 indexed channelId,
address indexed sender,
address indexed receiver,
uint256 value,
address tokenContract);
event Deposit(bytes32 indexed channelId,
uint256 deposit);
event Claim(bytes32 indexed channelId);
event StartSettling(bytes32 indexed channelId);
event Settle(bytes32 indexed channelId);

bytes32 public constant BL_ROLE = keccak256("BL_ROLE");

function init(
address _ERC20Address,
address _channelAddress,
uint256 _price) public initializer {
    _owner = msg.sender;
    ERC20Address = _ERC20Address;
    channelAddress = _channelAddress;

    __AccessControl_init_unchained();
```

```
        AccessControlUpgradeable._grantRole(
        DEFAULT_ADMIN_ROLE,
        _owner);

        token = IERC20Upgradeable(_ERC20Address);
        price = _price;
    }

    modifier isAdmin(address account){
        require(
        hasRole(
        DEFAULT_ADMIN_ROLE, account),
        "Controller: you have not permission for this query,
        u need to be Admin");
        _;
    }



    function open(
    bytes32 channelId,
    address receiver,
    uint256 settlingPeriod,
    uint256 tokenValue) public nonReentrant {
        require(
        isAbsent(channelId),
        "Channel with the same id is present");

        require(
        token.transferFrom(
        msg.sender,
```

```
        address(this),
        value),
        "Unable to transfer token to the contract");

        channels[channelId] = PaymentChannel({
            sender: msg.sender,
            senderDeposite: false,
            receiver: receiver,
            tokenValue: tokenValue,
            settlingPeriod: settlingPeriod,
            settlingUntil: 0
        });

        emit Open(
        channelId,
        msg.sender,
        receiver,
        value,
        tokenContract);
    }

    function deposite(bytes32 channelId) public nonReentrant {
        require(
        channels[channelId].sender == msg.sender,
        "You cant deposite funds");

        require(
        token.transferFrom(
        msg.sender,
        address(this),
        channels[channelId].value),
```

```
        "Unable to transfer token to the contract");
        channels[channelId].senderDeposite = true;


        emit Deposit(channelId, msg.sender);
    }

    function startSettling(bytes32 channelId) public {
        require(
        channels[channelId].senderDeposite == true,
        "sender did not deposite");

        PaymentChannel storage channel = channels[channelId];
        channel.settlingUntil =
        block.number.add(channel.settlingPeriod);

        emit StartSettling(channelId);
    }

    function canClaim(
    bytes32 channelId,
    uint256 payment,
    address origin,
    uint256 electricitySpend,
    bytes signature) public view returns(bool) {

        PaymentChannel storage channel = channels[channelId];
        bool isReceiver = origin == channel.receiver;
        require(
        payment <= channel.tokenValue,
        'payment more then tokenValue');
```

```
        bytes32 hash = recoveryPaymentDigest(
        channelId,
        payment,
        electricitySpend,
        channel.tokenContract);
        bool isSigned = channel.sender ==
        ECRecovery.recover(hash, signature);

        return isReceiver && isSigned;
    }

    function claim(
    bytes32 channelId,
    uint256 payment,
    uint256 electricitySpend,
    bytes signature) public nonReentrant {
        require(
        canClaim(
        channelId,
        payment,
        msg.sender,
        electricitySpend,
        signature),
        "canClaim returned false");

        PaymentChannel storage channel = channels[channelId];

        uint256 withdraw = channel.tokenValue - payment;


        require(
```

```
            token.transferFrom(
            address(this),
            channel.sender,
            withdraw),
            "Unable to transfer token to the contract");
            require(
            token.transferFrom(
            address(this),
            channel.receiver,
            payment),
            "Unable to transfer token to the contract");

            delete channels[channelId];

            emit DidClaim(channelId);
        }
}
```

A.2 Атака повторного використання на реалізований смарт-контракт

```
pragma solidity ^0.8.0;

import "contracts/diploma/controller.sol";

contract ReentrancyAttack {
    controller public vulnerableAddress;
    bytes32 channelId;
    uint256 payment;
    uint256 electricitySpend;
```

```
bytes signature;

constructor(
    address _vulnerableAddress,
    bytes32 _channelId,
    uint256 _payment,
    uint256 _electricitySpend,
    bytes _signature) {
    vulnerableAddress = controller(_vulnerableAddress);
    channelId = _channelId;
    payment = _payment;
    electricitySpend = _electricitySpend;
    signature = _signature;
}

fallback() external payable {
    if (address(depositFunds).balance >= 1 ether) {
        controller.claim(
        channelId,
        payment,
        electricitySpend,
        signature);
    }
}

function attack() external payable {
    controller.claim(
    channelId,
    payment,
    electricitySpend,
    signature);
```

```
        }


    }
```