МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Система виявлення та запобігання фішинговим атакам в однорангових мережах»

Виконавець: Олексій ОСАДЧИЙ

Керівник: д.т.н., доцент Аліна САВЧЕНКО

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*
Кафедра *комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«___»_____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Осадчого Олексія Васильовича
(ПІБ випускника)

1. Тема роботи: «Система виявлення та запобігання фішинговим атакам в однорангових мережах» затверджена наказом ректора № 623/ст від 01.05.2023р.
2. Термін виконання роботи: з 15 травня 2023 року по 25 червня 2023 року.
3. Вихідні дані до роботи: розробка аналітичної системи виявлення та запобігання фішинговим атакам в одноранговій мережі.
4. Зміст пояснювальної записки: 1. Аналітичний огляд і постановка задачі. 2. Технології та інструменти створення і використання смарт-контрактів. 3. Технологія створення програмного забезпечення для аналізу та прийому даних.
5. Перелік обов'язкового ілюстративного матеріалу: 1. Загальна схема роботи однорангових мереж. 2. Структура блокчейн блоку. 3. Структура бінарного дерева хешування. 4. Демонстрація взаємозв'язоку між взаємозамінними токенами та реєстром. 5. Вихідна структура даних. 6. Файлова структура проєкту. 7. Асинхронні HTTP-запити. 8. Графік атак на гаманці користувачів.

Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Огляд та аналіз предметної області. Написання 1 розділу, представлення керівнику.	15.05.2023-22.05.2023	
2	Вибір та опис використаних технологій. Написання 2 розділу, представлення керівнику.	23.05.2023-30.05.2023	
3	Написання 3 розділу, представлення керівнику.	31.05.2023-06.06.2023	
4	Загальне редагування та друк пояснювальної записки.	06.06.2023-08.06.2023	
5	Проходження нормоконтролю, перепліт пояснювальної записки.	09.06.2023-14.06.2023	
6	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	15.06.2023-19.06.2023	

7. Дата видачі завдання 15.05.2023р.

Керівник кваліфікаційної роботи _____ Аліна САВЧЕНКО
(підпис керівника)

Завдання прийняв до виконання _____ Олексій ОСАДЧИЙ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Система виявлення та запобігання фішинговим атакам в однорангових мережах»: містить: 66 сторінок, 10 рисунків, 22 інформаційні джерела, 1 додаток.

Об’єкт дослідження: система обміну даних в одноранговій мережі.

Предмет дослідження: автоматизація системи аналізу даних та процесу виявлення фішингових загроз.

Мета роботи: розробити систему виявлення та протидії фішинговим атакам та підвищення продуктивності аналізу в одноронговій мережі.

Методи дослідження, технічні та програмні засоби: розробка системи аналізу за допомогою Python і Debank API, перенаправлених запитів на основі проксі-серверів, порівняльний аналіз, обробка літературних джерел.

Отримані результати та їх новизна: розроблено систему виявлення та аналізу даних в одноранговій мережі з асинхованими функціями та стуктурованими результатами.

Матеріали кваліфікаційної роботи можуть бути використані при розробці систем аналізу в інших он-чейн мережах.

ОДНОРАНГОВА МЕРЕЖА, АНАЛІЗ ДАНИХ, СМАРТ-КОНТРАКТ, АДРЕС, DEBANK, ETHEREUM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

P2P(peer-to-peer)	−	Однорангові мережі
Web3	−	Децентралізований інтернет
DeFi	−	Децентралізовані фінанси
Dapps	−	Децентралізовані додатки
API	−	інтерфейс прикладного програмування
CSV file	−	Файл табличного зразку
EVM	−	Віртуальна машина Ethereum
L2(Layer2)	−	Блокчейн другого масштабування
NFT	−	Невзаємозамінний токен
EIP	−	Пропозиція щодо покращення Ethereum
URI	−	Уніфікований ідентифікатор ресурсів.

ВСТУП

Однорангові мережі є важливим інструментом для даних між вузлами без централізованого сервера. Проте, вони також піддаються ризикам і вразливостям, зокрема фішинговим атакам, які загрожують безпеці та конфіденційності даних користувачів

Метою даної роботи є розробка системи виявлення та протидії фішинговим атакам в одноранговій мережі та підвищення продуктивності процесу аналізу даних. Для досягнення цієї мети використовуються методи дослідження, такі як порівняльний аналіз і обробка літературних джерел. Технічні та програмні засоби включають розробку системи аналізу за допомогою мови програмування Python та використання Debank API для отримання необхідних даних. Для забезпечення безпеки та конфіденційності використовуються перенаправлення запитів через проксі-сервери.

Об'єктом дослідження даної роботи є процес обміну даними в одноранговій мережі, де важлива роль відводиться забезпеченню безпеки та запобіганню фішинговим атакам.

Предмет дослідження: автоматизація системи аналізу даних та процесу виявлення фішингових загроз.

Практичне значення: впровадження розробленої системи аналізу даних та процесу виявлення фішингових загроз повинна забезпечувати виявлення та протидію фішинговим атакам, аналізувати дані, забезпечувати безпеку мережі та підвищувати продуктивність процесу аналізу даних.

Розроблена система допоможе аналізувати великі групи даних, забезпечить безпеку в одноранговій мережі, підвищить продуктивність аналізу даних, збільшить ризик виявлення фішингової загроз.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАДАЧІ

У сучасному цифровому світі однорангові мережі є важливим інструментом для обміну даними між різними вузлами без централізованого сервера. Проте, вони також стають предметом ризиків і вразливостей, зокрема фішингових атак, які загрожують безпеці та конфіденційності даних. У зв'язку з цим розроблена аналітична система виявлення та протидії фішинговим атакам в однорангових мережах, що має велике значення для забезпечення безпеки даних та надійності мережі. Розроблена система допомагає аналізувати великі групи даних, забезпечує безпеку в одноранговій мережі та збільшує ризик виявлення фішингової загрози.

Метою даної роботи є розробка системи, яка автоматизує аналіз даних та процес виявлення фішингових загроз в одноранговій мережі. Для досягнення цієї мети використовуються методи дослідження, порівняльний аналіз і обробка літературних джерел. Технічні та програмні засоби включають розробку системи аналізу за допомогою мови програмування Python та використання Debank API для отримання необхідних даних. Забезпечення безпеки та конфіденційності передбачає перенаправлення запитів через проксі-сервери.

1.1. Однорангові мережі та їх вразливості

Однорангові мережі (Peer-to-Peer, P2P) є формою мережевої комунікації, в якій комп'ютери з'єднані безпосередньо між собою, утворюючи децентралізовану мережу. У такій мережі кожен комп'ютер може функціонувати як клієнт або сервер, відправляючи та отримуючи дані безпосередньо від інших комп'ютерів в мережі.

Більшість інформації, що передається в Інтернеті, використовує протокол зв'язку сервер/клієнт, де один сервер обслуговує запити декількох комп’ютерів. Ці комп’ютери є ідентифікаторами, які може розпізнати сервер, що забезпечує безпечну транзакцію.

На відміну від мереж сервер/клієнт, однорангові мережі складаються з вузлів або користувачів, які поводяться як сервер і клієнт, здатні одночасно запитувати інформацію і надавати її іншим вузлам.

Вузли були влучно названі серверами (Servents), взявши половину слова "сервер" і половину слова "клієнт". Вузли в децентралізованій мережі можуть не лише надсилати та отримувати дані, але й спільно використовувати такі ресурси, як обчислювальні потужності, сховища, периферійні пристрої та пропускну здатність мережі.

Однорангові мережі стали популярними після появи Napster, ранньої програми, яка дозволяла користувачам обмінюватися музикою з іншими користувачами. Він використовував гібридну мережу, де існував центральний сервер, який дозволяв користувачам шукати музику, а потім кілька користувачів вносили певний відсоток данних від файлу, який вони шукали.

Ця модель зручна своєю класичною можливістю пошуку файлів по всій мережі за допомогою напівцентралізованого сервера; однак, як буде показано далі, будь-яка форма централізованого групування вузлів може зробити всю мережу вразливою до атак. Поряд з гібридними одноранговими мережами існують також "чисті" однорангові мережі, в яких немає жодного централізованого вузла, а вся передача файлів відбувається між користувачами.

Однорангові мережі зазнають додаткових атак, таких як забруднення даних, що впливає на QoS(Quality of service), або відключення передачі даних за допомогою атак Eclipse, Sybil. Це відбувається шляхом атаки на субстрат маршрутизації мережі, який зазвичай базується на розподілених хеш-таблицях.

Вивчення P2P-мереж становить особливий інтерес, оскільки багато в чому, за відсутності централізованого контролю, вони імітують і навіть підсилюють соціальну поведінку. Ті ж проблеми, що стоять перед цілісністю інформації в мережах знань, проявляються і в однорангових мережах. Використання випадкових послідовностей для протоколів аутентифікації та виявлення зловмисників є важливим для захисту соціальної взаємодії.

1.2. Однорангова мережа криптографічного шифрування

Криптографічна однорангова мережа - це децентралізована мережева архітектура, в якій учасники, відомі як вузли, взаємодіють безпосередньо один з одним без потреби в центральному органі або посереднику. У контексті технології блокчейн, однорангова мережа, дозволяє декільком вузлам з'єднуватися і обмінюватися інформацією один з одним, формуючи розподілений реєстр.

У мережі блокчейн кожен учасник (вузол) зберігає копію всього блокчейну, який є хронологічним і незмінним записом транзакцій або даних. Коли відбувається нова транзакція або введення даних, вона поширюється на всі вузли мережі. За допомогою механізму консенсусу, такого як підтвердження роботи або підтвердження частки, вузли колективно погоджують дійсність і порядок транзакцій, забезпечуючи цілісність блокчейну.

Однорангова природа блокчейну надає кілька переваг, включаючи децентралізацію, прозорість, безпеку та стійкість. Вона усуває потребу в центральному органі влади, що ускладнює контроль або маніпулювання системою з боку одного суб'єкта. Транзакції перевіряються і підтверджуються

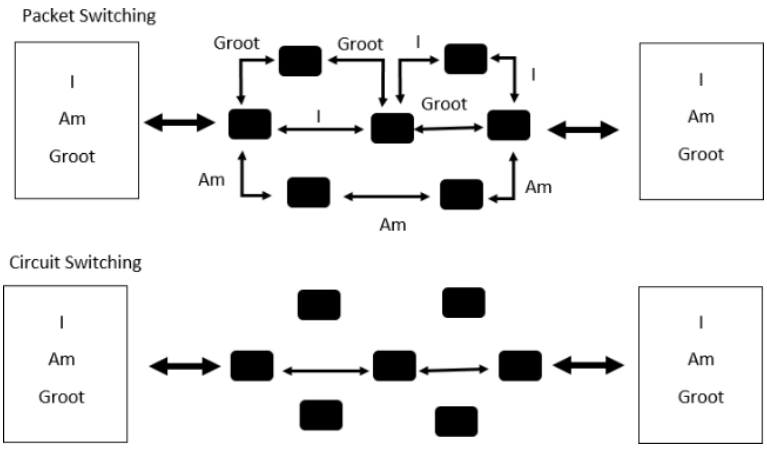
кількома вузлами, що підвищує довіру і знижує ризик шахрайства. Крім того, розподіленість мережі підвищує стійкість, оскільки вихід з ладу окремих вузлів не ставить під загрозу загальну функціональність блокчейну.

1.3. Історія розвитку однорангових мереж

Пірінгові мережі розпочалися з ARPANET, проект DARPA, започаткував пірінгові мережі в 1966 році. Ларрі Робертс у 1969 році успішно з'єднав комп'ютери в університетах UCLA, SRI, UCSB і UoU, використовуючи "комутацію пакетів". ARPANET дозволяла обмінюватися електронною поштою, файлами і здійснювати віддалений доступ. Вона починалася як експеримент, але постійно фінансувалася через військові інтереси.

ARPANET використовувала комутацію пакетів для передачі інформації між вузлами мережі. Інформація розбивалася на пакети, які включали повідомлення про призначення для маршрутизації. Цей метод подібний до сучасної мережі P2P, де інформація пересилається між вузлами, щоб потрапити до правильного місця, і мережа зазвичай є децентралізованою.

У другій версії ARPANET були додані протоколи TCP/IP для з'єднання вузлів у віртуальну мережу, незалежно від їхньої апаратної конфігурації.
ARPANET стимулювала створення мережі NSFnet, яка об'єднала 2000 університетських комп'ютерів. Ці мережі були попередниками сучасного



Інтернету.

Рис. 1.1. Комутація пакетів проти комутації каналів

Napster - відомий файлообмінний сервіс, запущений у 1999 році, що став популярним на початку Інтернету. Він отримав назву на честь Шона Паркера, одного з його засновників. Головною функцією Napster було надання користувачам можливості обмінюватися музичними файлами через Інтернет за допомогою піринговой мережі, що дозволяла прямий обмін файлами між комп'ютерами користувачів. За допомогою Napster люди могли знаходити, завантажувати та надсилати музику безкоштовно.

Проте Napster потрапив під судові позови музичних компаній, які заявляли, що сервіс допомагає порушувати авторські права та незаконно заробляти на музиці. У 2001 році суд прийняв рішення про заборону Napster та вимагав його припинити роботу.

Після закриття Napster з'явилися інші файлообмінні сервіси, такі як Kazaa, LimeWire, BitTorrent та Gnutella, які також стикалися з юридичними проблемами та були заборонені в деяких країнах. Однак файлообмінні сервіси існують й досі, проте з обмеженнями та захистом авторських прав, що дозволяє користувачам здійснювати легальний обмін файлами.

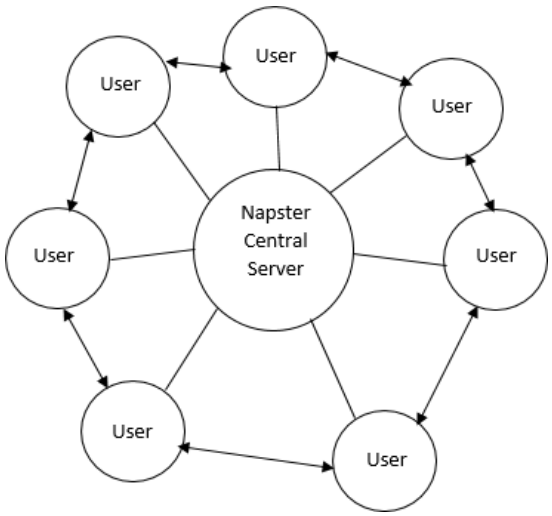
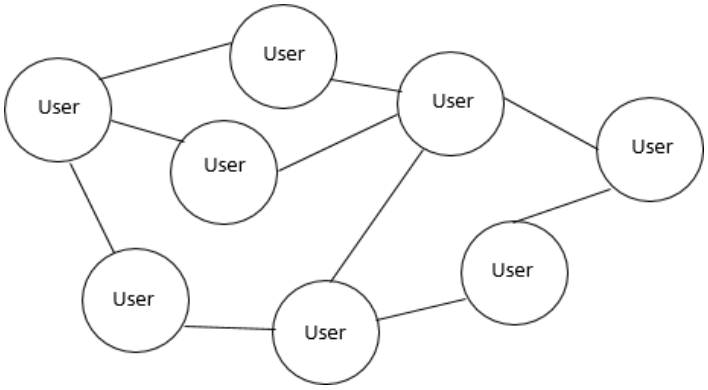


Рис. 1.2. Централізована P2P-архітектура Napster

Gnutella - пірингова файлообмінна мережа, запущена у 2000 році. Користувачі з'єднані у безцентральну мережу, де кожен може бути клієнтом і сервером одночасно. Мережа дозволяє пошук та завантаження файлів, що зберігаються на комп'ютерах інших користувачів. Gnutella не залежить від



центрального сервера, що робить її стійкішою до збоїв. Однак, мережа має вразливості, включаючи розповсюдження шкідливих файлів та обмежену пропускну здатність для великих файлів.

Рис. 1.3. Децентралізована P2P-архітектура Gnutella

BitTorrent - протокол для обміну файлами через Інтернет, створений у 2001 році. Він розбиває файли на дрібні частини для передачі між користувачами. Користувачі отримують торрент-файл з інформацією про файл і підключаються до мережі, одночасно завантажуючи його частини разом з іншими користувачами. Це забезпечує швидкий та ефективний обмін файлами. BitTorrent використовується як для легального обміну файлами, так і для незаконного обміну авторськими матеріалами, що має правові наслідки. Заходи безпеки включають перевірку файлів на віруси та використання офіційних джерел для отримання вмісту.

Bitcoin - криптовалюта, запропонована в 2008 році в документі "Bitcoin: A Peer-to-Peer Electronic Cash System" Сатоші Накамото. Метою було створити децентралізовану та прозору електронну грошову систему, яка не залежить від центральних установ. Біткоїн базується на технології блокчейну та використовує криптографію для підтвердження транзакцій і запобігання

подвійного витрачання. Учасники мережі погоджуються щодо стану та власності мережевих активів, а всі транзакції перевіряються майнерами для досягнення консенсусу та запобігання шахрайству.

1.4. Призначення технології блокчейн

У деяких країнах криптовалюти перебувають у правовому вакуумі, що означає, що їх статус чітко не визначений: їх використання і обіг не є офіційними, але вони також не заборонені. Це призводить до того, що операції з криптовалютами здійснюються "тіні".

Схожий статус має і технологія блокчейн. Вона також перебуває у правовому вакуумі, але існує важлива відмінність: блокчейн - це не гроші, контроль за виробництвом яких здійснює держава. Блокчейн - це технологія, яку можуть використовувати як державні структури, так і громадяни. Вона представляє собою електронну книгу обліку, яка зберігає інформацію про транзакції (в тому числі грошові операції) і є прозорою розподіленою базою даних.

У випадку біткоїна, транзакціями є грошові перекази між гаманцями користувачів, які забезпечують приватність. У контексті електронного уряду такі транзакції становлять результат взаємодії громадян і держави.

Принципи роботи блокчейн-системи наступні:

- Нові транзакції розсилаються всім вузлам.
- Кожен вузол збирає прийняті транзакції в блок.
- Кожен вузол намагається знайти хеш блоку, що задовольняє поточну складність.
- Як тільки хеш знайдений, блок відправляється в мережу.
- Вузли приймають блок, якщо всі транзакції в ньому коректні і не використовують вже витрачені кошти.
- Вузли виражають згоду з новими даними, починаючи працювати над наступним блоком і використовуючи хеш попереднього блоку як вихідні дані.

1.5. Безпека децентралізованої мережі

Однорангові мережі мають певні вразливості та проблеми з безпекою, які варто враховувати, коли використовується цей тип мережі. Декілька проблем безпеки однорангових мереж:

− Атаки на вузол мережі: Коли користувачі підключаються до однорангової мережі, вони стають вузлами мережі, які можуть бути піддані різного роду атакам, таким як атаки на вузли або підбір автентифікаційного коду. Недостатній рівень захисту може призвести до викрадення даних, а також може дозволити зловмисникам створювати фальшиві вузли, що ведуть до пошкоджень мережі.

− Шкідливі файли: Однорангові мережі можуть бути використані для обміну шкідливими файлами, такими як віруси, трояни або тими, що містять шпигунський вміст. Ці файли можуть поширюватися швидко та незалежно від волі користувачів, що може призвести до значних проблем з безпекою.

− Атаки на протокол: Однорангові мережі базуються на протоколах, які використовуються для забезпечення з'єднання між вузлами мережі. Якщо протокол мережі є вразливим, зловмисники можуть використовувати цю вразливість для атак на мережу та підключення до неї зловмисних вузлів.

− Використання ботнетів: Однорангові мережі можуть бути використані зловмисниками для створення ботнетів - мереж зражених комп'ютерів, які можуть використовуватися для здійснення атак на інші мережі або комп'ютери.

− Недостатня конфіденційність даних: Оскільки однорангові мережі базуються на співпраці та обміні даними між користувачами, інформація, яка передається в мережі, може бути доступна для інших користувачів, що може призвести до порушення конфіденційності даних. Якщо не застосовуються необхідні заходи захисту, то інформація може бути доступна для стеження, перехоплення або навіть крадіжки.

1.6. Структура шифрування блокчейну

Криптографічні хеш-функції відіграють вирішальну роль у забезпеченні цілісності та безпеки даних у різних додатках, включаючи технологію блокчейн. Хеш-функція - це математичний алгоритм, який отримує вхідні дані (дані будь-якого розміру) і видає результат фіксованого розміру, відомий як хеш або дайджест. Ключовою характеристикою криптографічної хеш-функції є те, що вона є детермінованою, тобто однакові вхідні дані завжди дають однаковий хеш-вихід.

В контексті блокчейну хеш-функції використовуються для створення унікальних відбитків блоків і транзакцій. Кожен блок містить хеш заголовка попереднього блоку, створюючи ланцюжок блоків, пов'язаних між собою. Цей механізм ланцюжка забезпечує незмінність блокчейну, оскільки будь-яка модифікація даних блоку змінює його хеш, розриваючи ланцюжок і роблячи підроблений блок і наступні блоки недійсними.

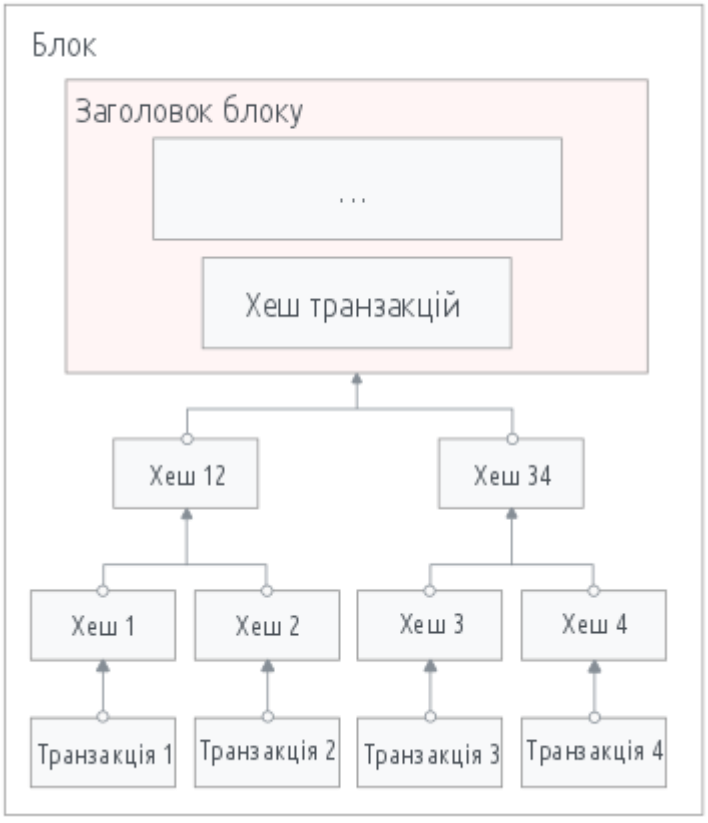


Рис. 1.4. Структура блокчейн блоку

Крім того, окремі транзакції всередині блоку хешуються для створення дерева Меркла або структури хеш-дерева. Це дерево дозволяє ефективно перевіряти цілісність всього блоку. Порівнюючи хеш кореневого вузла (корінь Меркла) з відповідним значенням, що зберігається в заголовку блоку, учасники можуть швидко визначити, чи була змінена будь-яка транзакція в блоці. [1]

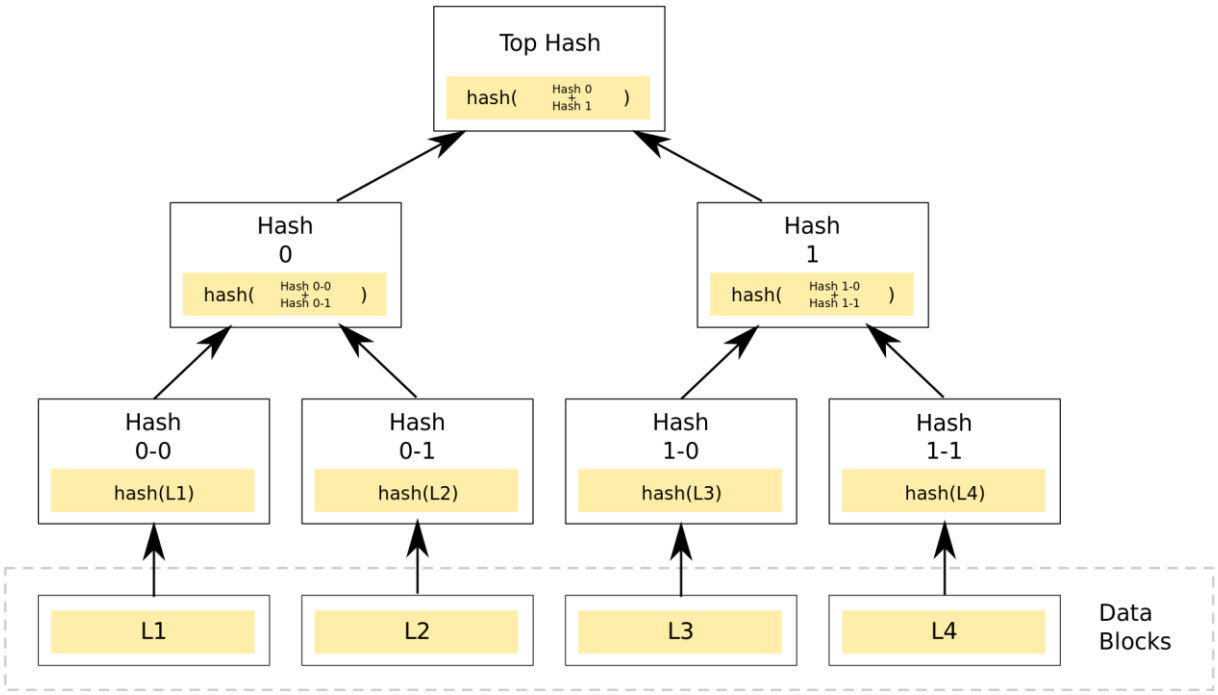


Рис. 1.5. Приклад бінарного дерева хешування

Криптографічні хеш-функції забезпечують високий рівень безпеки завдяки кільком важливим властивостям. По-перше, вони обчислювально ефективні, що дозволяє швидко хешувати дані. По-друге, вони створюють унікальні хеші для різних вхідних даних, гарантуючи, що навіть незначна зміна даних призведе до зовсім іншого хеш-значення. По-третє, вони незворотні, тобто практично неможливо відновити початкові вхідні дані лише за хеш-значенням.

Використання криптографічних хеш-функцій в технології блокчейн забезпечує надійний і стійкий до підробки механізм забезпечення цілісності даних. Покладаючись на властивості хеш-функцій, мережі блокчейн можуть встановлювати довіру між учасниками, забезпечуючи безпечні та прозорі транзакції і підтримуючи цілісність розподіленого реєстру.

1.7. Конфіденційність та ідентичність в одноранговій мережі

Конфіденційність та ідентичність у криптовалютах та блокчейні - важливі питання. Криптовалюти, такі як біткоїн, забезпечують псевдоанонімність, але можуть виникати проблеми з конфіденційністю. Використання криптовалют Monero і Zcash дозволяє покращити конфіденційність. Управління ідентифікацією також є важливою частиною. Децентралізовані рішення, такі як децентралізовані ідентифікатори (DID) і перевірені облікові дані, дозволяють більше контролю над особистою інформацією. Важливо збалансувати конфіденційність та ідентичність з дотриманням нормативних вимог безпеки. Користувачі повинні дотримуватися правил KYC та AML. Забезпечення правильного балансу між конфіденційністю, безпекою та дотриманням нормативів - важливе завдання для криптовалют та захисту прав користувачів.

1.8. Види фішингових атак

Основна мета інформаційної безпеки полягає в захисті конфіденційної інформації від фішингових атак, які є серйозною загрозою для організацій та користувачів. Фішинг - це процес соціальної інженерії, коли зловмисники маніпулюють людьми, щоб отримати доступ до конфіденційної інформації або досягнути неправомірної вигоди.

Фішингові атаки стають все поширенішими через зростаючу кількість доступних інтернет-сервісів. Зловмисники використовують різні методи, такі як фейкові повідомлення про виграш призів або фейкові електронні листи,

щоб отримати доступ до конфіденційної інформації користувачів. Фішинг може призвести до фінансової шкоди, особливо в контексті криптовалют, де зловмисники намагаються отримати приватні ключі або доступ до криптовалютних активів користувачів.

Для захисту від фішингових атак, користувачам необхідно бути пильними і дотримуватися найкращих практик безпеки. Це включає перевірку автентичності веб-сайтів, використання двофакторної автентифікації і ніколи не передавати конфіденційну інформацію або приватні ключі. Крім того, криптовалютні платформи повинні вживати заходів безпеки, щоб запобігти фішинговим атакам, і забезпечити навчання користувачів про ризики фішингу.

Фішингові атаки в сфері невзаємозамінних токенів та криптовалют мають свої особливості. Зловмисники використовують шахрайські схеми, щоб отримати доступ до приватних ключів або гаманців користувачів. Вони можуть видаватися представниками популярних NFT-маркетплейсів або використовувати інші хитроманіпуляції. Захист від цих атак полягає в уважності, перевірці автентичності та усвідомленні ризиків фішингу.

В цілому, свідомість про фішингові атаки, дотримання найкращих практик безпеки і спільна відповідальність платформ та користувачів допоможуть знизити ризик фішингу та забезпечити безпеку конфіденційної інформації та криптовалютних активів.

1.9. Постановка задачі

В епоху інформаційних технологій активно відбувається процес зростання кількості фішингових атак, спрямованих на користувачів однорангових мереж. Фішинг є методом шахрайства, при якому зловмисники намагаються отримати конфіденційну інформацію, в даному випадку, таку як seed-фрази; підпис сторонніх смарт-контрактів або заплутати і збити з пантелику користувача, шляхом підробки інформації, яка надходить від довірених особ. Для захисту користувачів та попередження фішингових атак

в однорангових мережах, стає актуальною задача розробки системи виявлення та запобігання фішинговим атакам.

Мета даної дипломної роботи полягає у створенні системи, яка забезпечуватиме аналіз та виявлення фішингових атак в однорангових мережах і запобігатиме їх наслідкам. Дана система буде здатна реагувати аналізувати адреси гаманців, ідентифікувати затверджені та підроблені елементи, виявляти характерні ознаки фішингу та уникати його.

Для досягнення цієї мети необхідно вирішити такі завдання:

Провести огляд і аналіз існуючих методів та алгоритмів виявлення фішингових атак в однорангових мережах. Врахувати різні види фішингу та їх характеристики.

− Розробити системи аналізу та виявлення фішингових атак, що включатиме компоненти для моніторингу активності гаманців, виявлення підозрілих дій та реагування на фішингові атаки.

− Реалізувати необхідні алгоритми та інструменти, такі як аналіз затвердженних смарт-контрактів, перевірка отриманних даних, перехоплення мережевого трафіку, систему виявлення та ліквідування загроз.

− Провести експериментальне дослідження розробленої системи з використанням реальних гаманців. Оцінити ефективність системи у аналізв запобіганні фішинговим атакам.

− Успішна реалізація даної системи виявлення та запобігання фішинговим атакам в однорангових мережах буде сприяти захисту користувачів від шахрайства, збереженню їх конфіденційної інформації та підвищенню загального рівня безпеки у блокчейні.

ВИСНОВКИ ДО РОЗДІЛУ 1

Проведено огляд однорангових мереж, технології блокчейн та їх використання в сучасному світі. Підсумок однорангових мереж та технологій блокчейну вказує на їх широке використання у сучасному світі. Однорангові мережі дозволяють безпосередній обмін даними між комп'ютерами і спільне використання ресурсів.

Криптографічні однорангові мережі, засновані на блокчейні, надають децентралізовану структуру з високою прозорістю та безпекою. Втім, вони також мають вразливості, такі як атаки на вузли, поширення шкідливих файлів та фішингові атаки.

На сьогодні для забезпечення безпеки використовуються криптографічні протоколи, методи шифрування та механізми аутентифікації. Особливо небезпечні фішингові атаки, які можуть загрожувати конфіденційності та безпеці користувачів криптовалют.

Для захисту від них необхідно бути обережними та використовувати додаткові заходи безпеки, такі як перевірка автентичності веб-сайтів та використання двофакторної автентифікації.

В цілому, безпека є важливою складовою блокчейну, і для ефективного використання його потенціалу необхідно забезпечити високий рівень захисту і постійно підвищувати свідомість про безпеку мереж.

РОЗДІЛ 2
ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ СТВОРЕННЯ І ВИКОРИСТАННЯ СМАРТ-КОНТРАКТІВ

Технології та інструменти створення і використання смарт-контрактів прогресують швидкими темпами, розвиваючи нові можливості і забезпечуючи безпеку у сфері блокчейну. Одним з ключових аспектів є технології базування смарт-контрактів, які дозволяють розробникам вибирати серед різних платформ і блокчейнів для створення та розгортання контрактів. Це створює гнучкість і відкритість у виборі розробки та забезпечує сумісність з різними блокчейнами.

Ще одним важливим аспектом є технології аудиту смарт-контрактів, які допомагають перевірити код контракту на наявність помилок та вразливостей перед його виконанням. Це забезпечує надійність і безпеку контрактів, допомагає уникнути можливих критичних помилок та зловживань з боку зловмисників.

У сфері токенізації активів технології стандартів взаємозамінних токенів грають важливу роль. Ці стандарти визначають універсальний формат токенів, який дозволяє їх взаємодію з різними платформами і сервісами. Це спрощує процес взаємодії між різними проектами, сприяє ліквідності та підвищує широку прийнятність токенів.

Окремо варто згадати про технології стандартів невзаємозамінних токенів (NFT), які забезпечують унікальність та неповторність токенів. Ці технології дозволяють створювати токени, які представляють унікальні цифрові активи. NFT стають все більш популярними, викликаючи значний інтерес в сфері культури і інвестицій.

2.1. Технології базування смарт-контрактів

З моменту випуску біткоїна у 2009 році децентралізовані однорангові мережі, такі як блокчейн, стали популярними та набули широкого розповсюдження. Цей термін використовувався (і досі використовується) для позначення юридичного контракту, який повністю або, принаймні, його частини який може бути виражений та реалізований у програмному забезпеченні. Криптовалюта не керується централізованим органом, а базується на автоматизованому консенсусі між користувачами мережі. Протокол консенсусу використовується для підтримки та захисту публічних даних, доступних тільки для додавання до реєстру транзакцій. Сьогодні цей термін широко використовується для позначення кодових сценаріїв, які синхронно виконуються на декількох вузлах розподіленого реєстру.

Технологія блокчейн та децентралізованих мереж використовуються не тільки для криптовалют, але й для смарт-контрактів. Термін "смарт-контракт" використовувався спочатку для автоматизації юридичних контрактів загалом. В контексті блокчейну, смарт-контракт - це програмний код, що автоматично виконується на декількох вузлах розподіленої мережі без необхідності довіреної сторони.

Смарт-контракти мають властивість самостійного виконання і використовуються в різних сферах, де потрібні транзакції на основі даних. За останні роки популярність смарт-контрактів і блокчейну значно зросла. Є багато розробників, які спеціалізуються на розробці смарт-контрактів для різних блокчейнів, включаючи Ethereum та інші EVM-сумісні блокчейни, такі як Binance Smart Chain, Polygon, Avalanche та інші. Ці блокчейни надають розробникам альтернативні платформи з низькими комісіями та швидшими підтвердженнями транзакцій.

На початку 2018 року понад 10% вакансій, оголошених на Guru2, одному з найбільших сайтів для фрілансерів, були пов'язані зі смарт-контрактами та блокчейном.

На найбільш популярному блокчейні, на якому можна виконувати смарт-контракти, Ethereum, згідно зі звітом, кількість розгорнутих смарт-контрактів у 4 кварталі 2022 року зросла на 453% порівняно з попереднім кварталом, досягнувши приголомшливих 4.6 мільйона одиниць. [2-6]

Смарт-контракти - це самостійні угоди, які працюють в блокчейн мережах. Термін "смарт-контракт" був запропонований Ніком Сабо в середині 1990-х років. Вони є комп'ютерними програмами, що автоматично виконують заздалегідь визначені дії при виконанні певних умов. Вони дозволяють сторонам брати участь у транзакціях, не покладаючись на центральні сервери, знижуючи витрати та підвищуючи ефективність.

Смарт-контракти усувають потребу в посередниках і дозволяють здійснювати безпечні, прозорі та незмінні транзакції.

Ці контракти зберігаються в блокчейні, що забезпечує їхню незмінність і прозорість. Вони автономно працюють, виконуючи дії без необхідності людського втручання. Смарт-контракти дозволяють автоматизувати та оптимізувати складні процеси, знижуючи витрати та підвищуючи ефективність.

Ці контракти також підвищують довіру та безпеку, оскільки їхня працездатність та деталі записуються і підтверджуються учасниками мережі блокчейн. Децентралізований характер блокчейну забезпечує незмінність умов контракту і зміцнює довіру між сторонами.

Крім того, смарт-контракти дозволяють створювати децентралізовані додатки (DApps), які надають користувачам широкий спектр послуг і функцій без посередників. DApps знайшли застосування в різних галузях, включаючи фінанси, управління ланцюгами поставок, охорону здоров'я та системи голосування.

Проте, важливо враховувати, що смарт-контракти можуть мати вразливості або помилки в коді, що може призвести до серйозних наслідків.

Тому аудит та перевірка коду мають вирішальне значення для забезпечення надійності та безпеки смарт-контрактів.

Сьогодні смарт-контракти революціонізують спосіб укладання угод і мають потенціал трансформувати традиційні системи. З розвитком блокчейну, їхня роль стає ще більш значущою у майбутніх транзакціях і угодах.

Аудит смарт-контрактів важний для безпеки й надійності додатків на блокчейні. Він включає перевірку коду, дизайну та функціональності контракту, щоб виявити потенційні вразливості та слабкі місця. Аудит забезпечує довіру до системи та зменшує ризик фінансових втрат. Використовуються спеціалізовані інструменти та методології, аудит можуть проводити як внутрішні, так і зовнішні аудитори. Він відіграє важливу роль у розвитку смарт-контрактів та забезпечує надійність екосистеми блокчейну. [7-8]

2.2. Технології аудиту смарт-контрактів

Хороший аудит смарт-контрактів досягає двох ключових цілей: безпеки та довіри.

По-перш, гарантія якості коду смарт-контракту допомагає виявити потенційні проблеми та гарантує, що протокол вживає необхідних заходів для усунення будь-яких помилок або недоліків, які можуть поставити під загрозу користувачів. Хоча немає жодних гарантій, що протокол буде безпечним після аудиту, хороший аудитор смарт-контрактів все одно може виконувати комплексні перевірки, щоб виявити потенційні проблеми, потенційно запобігаючи катастрофічній вразливості після запуску.

По-друге, аудит допомагає проекту отримати певний рівень впевненості та довіри криптоспільноти, а також потенційних венчурних інвесторів. Надаючи гарантіїї щодо базового рівня безпеки. Це важливо не лише для нових проектів, які виходять на ринок, але й для існуючих проектів, які

розгортають серйозне оновлення. Перед розгортанням будь-яких серйозних змін у виробництві смарт-контрактів, аудиторська перевірка, проведена стороннім аудитором, швидко стає стандартною практикою.

Це передбачає певні перевірчі компанії також пропонувати послуги з кібербезпеки, такі як тестування на обхід правил коду, запуск баунті-програм винагород за помилки, оцінка вразливостей і моделювання загроз. Усі ці додаткові послуги проект може задіяти, якщо йому знадобиться допомога чи підтримка.

2.3. Технології стандартів взаємозамінних токенів

Стандарт ERC-20, запропонований Фабіаном Фогельстеллером в листопаді 2015 року, - це стандарт токенів, який реалізує API для токенів в рамках смарт-контрактів.

Стандарт токенів ERC-20 - це набір правил і рекомендацій, які визначають, як токени повинні створюватися і поводитися в блокчейні Ethereum. ERC-20 розшифровується як "Ethereum Request for Comments 20", що є стандартним ідентифікатором для цього стандарту токенів.

ERC-20 вводить стандарт для взаємозамінних токенів, іншими словами, вони мають властивість, яка робить кожен токен точно таким же (за типом і значенням), як і інший токен. Наприклад, токен ERC-20 діє так само, як і ETH(Нативний токен Ethereum), означає, що кожен токен ідентичний і може бути обміняний на основі "один до одного" з іншими токенами того ж типу, випущеним одним смарт-контрактом. Ці токени стали найбільш поширеним і підтримуваним типом токенів в мережі Ethereum, на них працює величезна кількість децентралізованих додатків (dApps) і токенізованих екосистем. Багато з цих dApps можуть підключатися і працювати разом для створення складних фінансових послуг.

Його широке використання можна пояснити стандартизацією, яку він забезпечує, дозволяючи розробникам створювати токени, сумісні з різними

гаманцями, біржами та іншими смарт-контрактами. Ця сумісність зробила токени ERC-20 стандартом де-факто для створення та обміну токенів в мережі Ethereum.

Токени ERC-20 поставляються з набором вбудованих функцій, які дозволяють розробникам реалізовувати різні можливості. Ці функції включають можливість передавати токени між адресами, перевіряти баланс токенів облікового запису і підтверджувати передачу токенів від імені іншого облікового запису. Дотримуючись цих загальних функцій, токени ERC-20 забезпечують узгодженість і простоту використання як для розробників, так і для користувачів.

Стандарт ERC-20 став основним для первинних пропозицій монет та продажу токенів на блокчейні Ethereum. Його стандартизована структура спрощує процес створення та розповсюдження токенів, полегшуючи проектам запуск власних кампаній зі збору коштів та забезпечення ліквідності токенів.

Універсальність токенів ERC-20 сприяла зростанню децентралізованих додатків (dApps) і токенізованих екосистем. Ці токени слугують фундаментальним будівельним блоком для широкого спектру додатків, включаючи децентралізовані фінанси (DeFi), ігри, не взаємозамінні токени (NFT) тощо. Стандарт ERC-20 сприяв створенню інноваційного середовища, що дозволяє розробникам створювати нові та унікальні варіанти використання технології блокчейн.

Хоча ERC-20 залишається найпоширенішим стандартом токенів на Ethereum, важливо відзначити, що спільнота Ethereum продовжує розвиватися і впроваджувати нові стандарти токенів. Серед них такі стандарти, як ERC-721 для невзаємозамінних токенів і ERC-1155 для контрактів з декількома токенами. Ці стандарти, що розвиваються, спрямовані на конкретні випадки використання і пропонують додаткову функціональність за межами ERC-20,

ще більше розширюючи можливості для токенізації в блокчейні Ethereum. [9-11]

2.4. Технології стандартів невзаємозамінних токенів(NFT)

Невзаємозамінний токен (NFT) - це унікальний цифровий ідентифікатор, який записується в блокчейні і використовується для підтвердження права власності та автентичності. Його неможливо скопіювати, замінити або розділити.

ERC-721 - це стандарт невзаємозамінних токенів (NFT), який надає стандартний API для NFT в рамках смарт-контрактів на блокчейні Ethereum. Він був створений Вільямом Ентрікеном, Дітером Ширлі, Джейкобом Евансом та Настасією Сакс у січні 2018 року. Стандарт ERC-721 дозволяє відстежувати і передавати унікальні токени, які можуть представляти право власності на цифрові або фізичні активи.

ERC-721 вирізняється своїм стандартним інтерфейсом, який дозволяє гаманцям, брокерам та аукціонним додаткам працювати з будь-якими NFT на Ethereum. У відміну від взаємозамінних токенів, де кожен токен ідентичний і взаємозамінний, NFT є відмінними і унікальними. Стандарт ERC-721 спрямований на відстеження прав власності та полегшення безпечної передачі цих унікальних активів.

ERC-1155 - це стандарт, який дозволяє управляти декількома типами токенів в рамках одного смарт-контракту. Він використовує ідентифікатори токенів для представлення конфігурованих типів токенів з власними метаданими, постачанням і атрибутами. Цей стандарт ефективно управляє токенами, зменшує надлишковий байт-код і підтримує передачу декількох типів токенів та торгівлю ними в одній транзакції.

ERC-1155 надає функції для безпечного переказу токенів, пакетного переказу, запитів балансу, управління затвердженням та сповіщення про події. Контракти, які реалізують ERC-1155, повинні відповідати цим функціям і

генерувати необхідні події. Також контракти, які хочуть отримувати токени ERC-1155, повинні реалізувати інтерфейс ERC1155TokenReceiver і підтримувати інтерфейс ERC-165.

ERC-1155 є гнучким і ефективним рішенням для управління декількома типами токенів в рамках одного контракту. Цей стандарт забезпечує стандартизований підхід для безпечної і стандартизованої передачі токенів, забезпечуючи безпеку і надійність транзакцій.

Порівнюючи ERC-1155 з ERC-721, є кілька ключових відмінностей. Токени ERC-721 представляють унікальні і неподільні активи, тоді як токени ERC-1155 можуть представляти як унікальні, так і кілька ідентичних активів в рамках одного контракту. Токени ERC-721 мають унікальні ідентифікатори і не можуть бути відтворені або замінені, тоді як токени ERC-1155 є як взаємозамінні, так і не взаємозамінні, надаючи більшу гнучкість в управлінні різними типами активів.

ERC-6551 - По суті ERC-6551 перетворює звичайні NFT в повноцінні облікові записи, які можуть володіти активами, взаємодіяти з даппами та служити засобом ідентифікації. Стандарт був запущений на мейннеті Ethereum 7 травня 2023 року. Причиною появи ERC-6551 стало зростаюче використання NFT для ончейн-ідентифікації.

ERC-6551 - це стандарт, який забезпечує сумісність з існуючими

ERC-721 шляхом розгортання унікальних облікових записів смарт-контрактів, відомих як "прив'язані до токену облікові записи" (Token Bound Accounts, TBA), для кожного токена ERC-721.

ERC-6551 дозволяє використовувати NFT для зберігання інших NFT, торгівлі ERC-20 токенами та іншими стандартними операціями з обліковим записом. Це дає можливість накопичувати NFT в Web3-іграх, замість зберігання їх у звичайному гаманці. За допомогою ERC-6551, їх можна зберігати в NFT на вибір, наприклад, у аватарі гравця. Це дозволяє в майбутньому продавати аватар та накопичені предмети як один NFT. Крім

того, NFT з власними обліковими записами та активами не можуть бути просто скопійовані правою кнопкою миші.

ERC-6551 також дозволяє NFT мати кілька адрес, прив'язаних до одного токена в різних мережах, що сприятиме новим експериментам та інноваціям в галузі NFT.

Варто зазначити, що ERC-6551 не підтримує NFT, створені до впровадження ERC-721, наприклад, оригінальні КриптоПанки. Крім того, стандарт ERC-6551 піднімає два важливих питання безпеки: уникнення шахрайської поведінки та запобігання безкінечним циклам володіння. Це означає, що ERC-6551 має вбудовані механізми для запобігання шахрайським схемам та недоступності активів на обліковому записі TBA. [12-15]

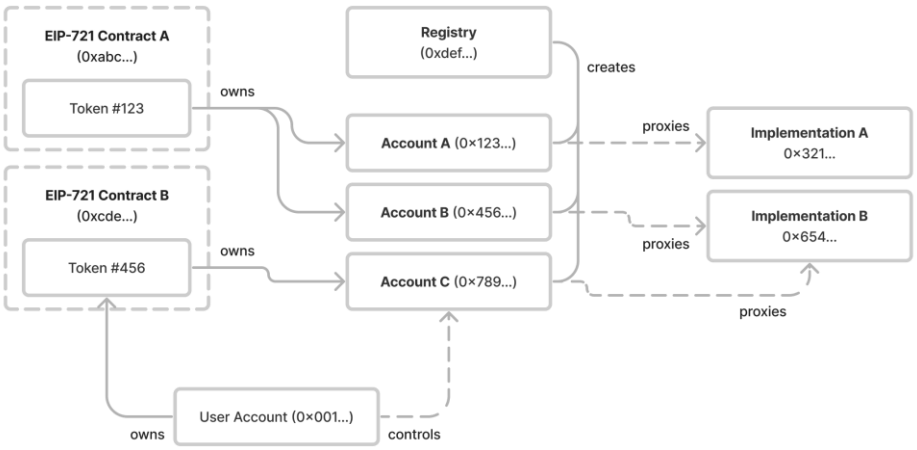


Рис. 2.1. Демонстрація взаємозв'язоку між токенами ERC-721, власниками токенів ERC-721, обліковими записами, прив'язаними до токенів, та реєстром.

2.5. Способи аналізу та моніторинг токенів

Etherscan.io - пошукова система для блокчейну Ethereum, що дозволяє користувачам відстежувати, перевіряти та досліджувати транзакції, смарт-контракти та адреси. Вона надає вичерпні дані про історію транзакцій, блоки, токени та взаємодію контрактів. Etherscan забезпечує прозорість та

підзвітність в екосистемі Ethereum. Проте, важливо бути обережним, оскільки фішингові транзакції, що містять шахрайські практики, можуть обдурити користувачів та призвести до втрати коштів. Ретельне дослідження та перевірка контрактів та токенів можуть допомогти уникнути ризиків фішингу в блокчейні Ethereum.

2.6. Дослідження фішингової отруйної атаки

Протягом останніх місяців спостерігається значне збільшення кількості атак з отруєнням адрес, спрямованих на те, що з гаманця виконується вихідна транзація на суму 0 busd/usdt. Атаки мають серйозні наслідки, оскільки було скомпрометовано понад 340 000 адрес у блокчейні за перші два тижня. Це призвело до того, що за перші дні 99 жертв втратили загалом понад 1 640 000 usdt або в еквіваленті, 1.64 мільйона доларів США. Користувачам, які зіткнулися з такою ситуацією, не варто панікувати, оскільки їхні активи в безпеці, а приватні ключі залишаються недоторканими.

В цьому розділі зробимо ретельний аналіз сценарію атак, включаючи ідентифікацію та відстеження зловмисників у блокчейні. Крім того, зробимо детальний аналіз методів, які використовували зловмисники для здійснення цих атак. Користувачі уже закликають розробників пошукових систем вдосконалити свої системи оповіщення про ризики взаємодії з цими токенами.

За допомогою ehterscan ми можемо виявити повторювану схему переказів на суму 0 busd/usdt в мережах Ethereum та Binance Smart Chain . Щоб проілюструвати це явище, розглянемо конкретну транзакцію в ланцюжку BSC в якості прикладу.

Рис. 2.2. Користувач надсилає 3000 bsc-usdt адресу із закінченням 451cb.

Зловмисник відстежує інформацію про транзакції стейблкоінів блокчейні і перехоплює інформацію про переказ, на яку адресу був здійснений переказ коштів.

Для здійснення атаки хакер ретельно підбирає адресу, яка має ті ж самі перші та останні цифри, що й адреса користувача. Структура виклику токенів передачає, що будь-який адрес може викликати у любого іншого адресу транзакцію з вмістом 0 busd/usdt. Це передбачає, ща хакер може ініціювати такого типу транзакції. Зловмисник може використовувати такі інструменти, як Profanity, інструмент генерації красивих адрес гаманця, щоб швидко згенерувати адресу з тими самими першими та останніми сімома цифрами, що й адреса користувача.

Рис. 2.3. Вхідна транзакція 0 usdt.

Зловмисник згенерував адресу із тим самим закінченням, що і адреса користувача, якому переказали токени. І здійснив транзакцію з вмістом 0 usdt на адрес відправника.

Наступного разу, коли жертва буде використовувати пошукові сервіси різного виду, як телеграм боти, або etherscan, скопіює підробний адрес зловмисника і здійснить переказ коштів, як у цьому випадку.

Рис. 2.4. Жертва атаки здійснила переказ на підробний адрес.

Дана атака називається атакою отруєнням, тому що вражає адреси користувачів, щоб при огляді через пошукові сервіси для дослідження транзакцій, адреса зловмисника з’являлася в історії транзакцій користувача, заманюючи користувача помилково прийняти її за довірену адресу, з якою проходила оригінальна взаємодія.

Крім того, хакер створює адресу, яка має ті ж самі перші та останні цифри, що й довірена адреса користувача, і використовує її для наступної транзакції.

Станом на сьогодні здійснюється від п’яти до сорока тисяч атак кажної години тільки на блокчейні Binance Smart Chain. Знаходячи інформацію на блокчейні Ethereum, ми бачимо від 50 до 200 атак щогодини.

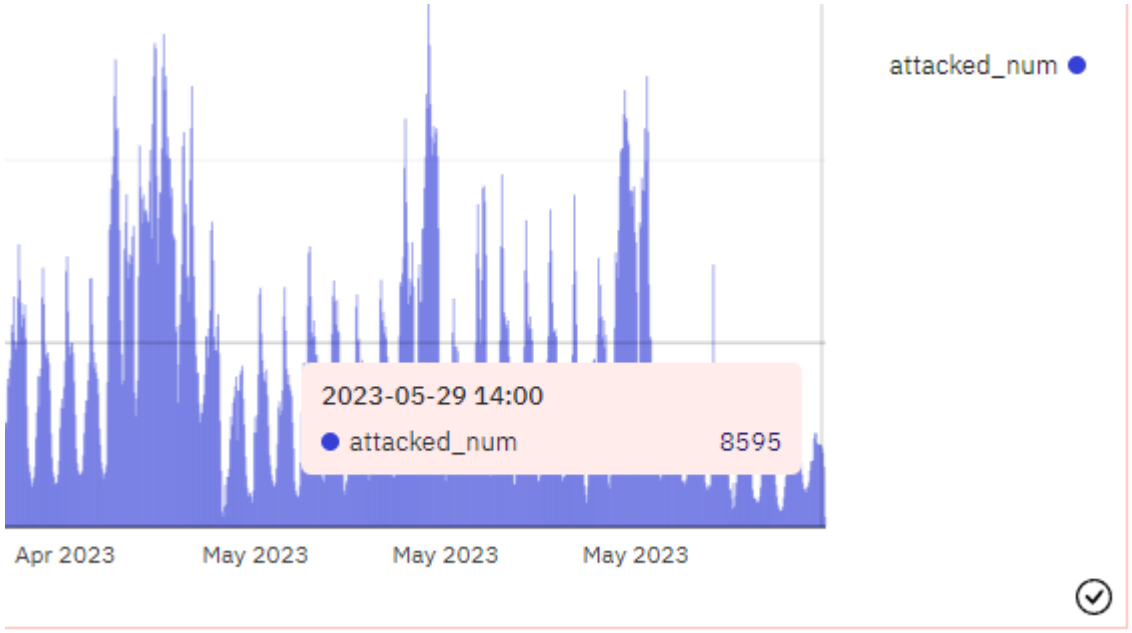


Рис. 2.5. Погодинний графік атак на блокчейні Binance Smart Chain.

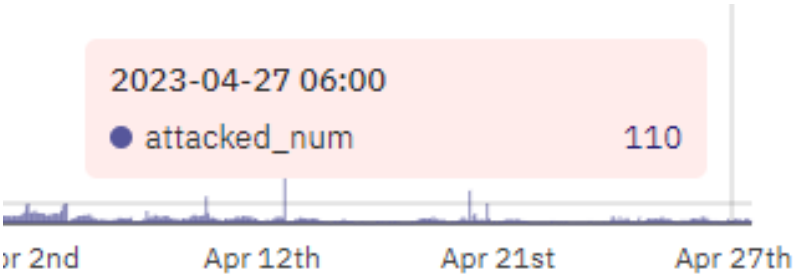


Рис. 2.6. Погодинний графік атак на блокчейні Ethereum.

Станом на 28 травня 2023 року кількість атак на блокчейн Binance Smart Chain перевищила 54 мільйонів спроб, а кількість незалежних адрес, які постраждали від атак, перевищила двісті тисяч!

Рис. 2.7. Загальна кількість атак на гаманці.

Якщо говорити про тенденції, то блокчейн BSC вибухає від атак з 22 листопада, а блокчейн ETH - з 27 листопада 2022 року, причому масштаби атак на обидва ланцюжка збільшуються.

Крім того, можна помітити, що існує значна закономірність у часі проведення атак, зі значним зменшенням обсягу атак між 17:00 UTC і 0:00 UTC кожного дня. Підозрювані зловмисники перебувають в азіатському часовому поясі. [16-21].

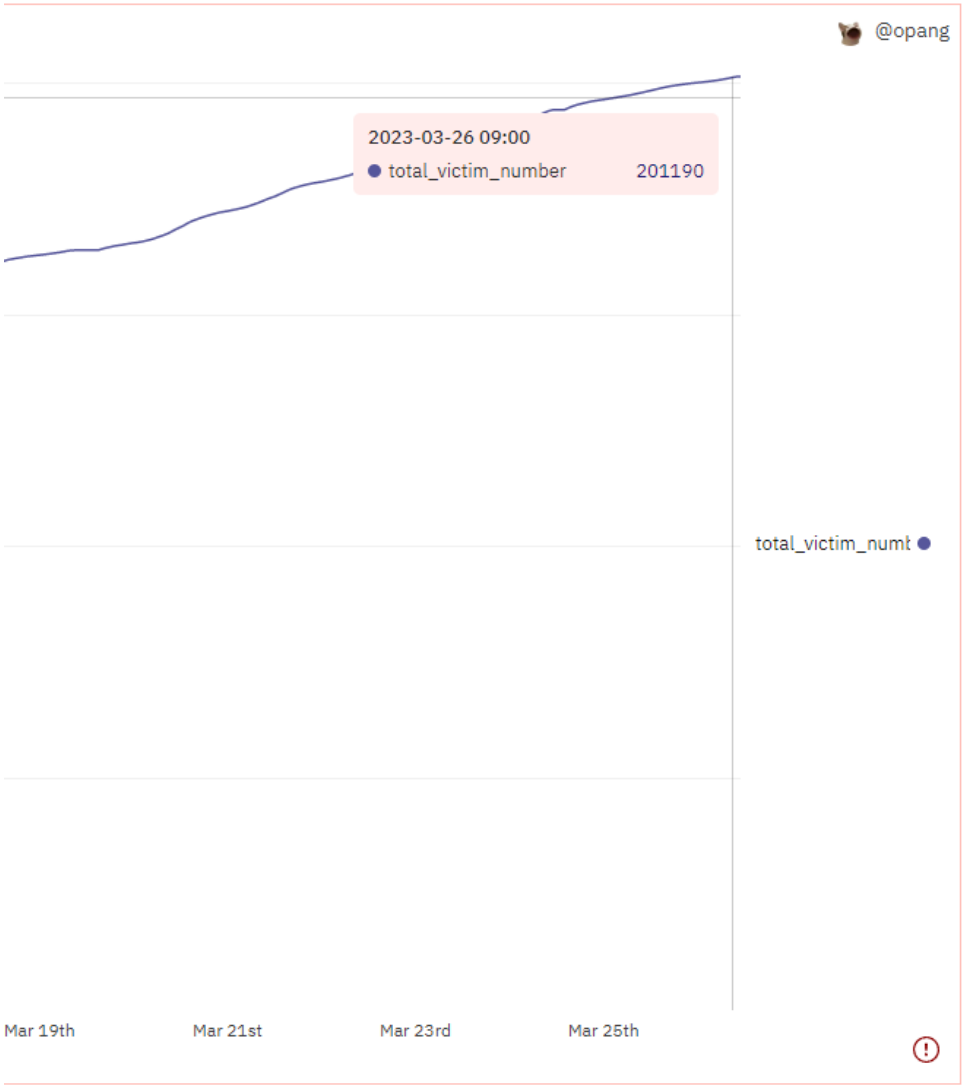


Рис. 2.8. Загальна кількість жертв.

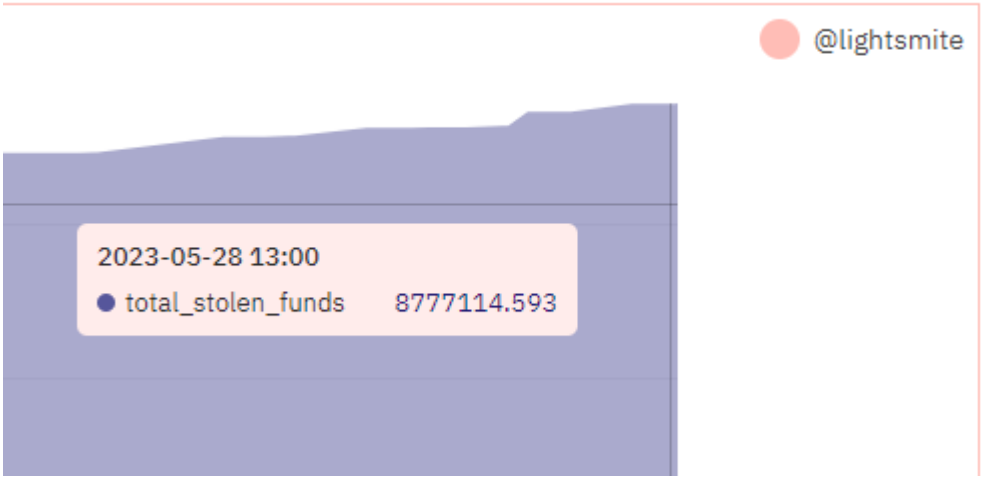


Рис. 2.9. Загальна кількість вкрадених коштів, еквівалентних 8.7 мільйонів доларів США.

ВИСНОВКИ ДО РОЗДІЛУ 2

Проведено огляд стандартів токенів в екосистемі Ethereum. Їх поява перевернула спосіб створення, управління та обміну цифровими активами. Чотири основні стандарти взаємозамінних і невзаємозамінних токенів у Ethereum - ERC-20, ERC-721, ERC-1155 і ERC-6551 - кожен з них виконує різні завдання і відповідає різноманітним сценаріям використання.

Токени ERC-20 стали стандартом для створення та обміну взаємозамінних токенів, що сприяло розвитку DeFi і децентралізованих додатків. Вони забезпечують взаємодію між різними гаманцями, біржами і смарт-контрактами.

Стандарт ERC-721 ввів поняття невзаємозамінних токенів (NFT) і знайшов застосування в цифровому мистецтві, іграх та колекціонуванні. Вони токенізують унікальні активи і спрощують перевірку власності та походження.

ERC-1155 пропонує ефективне управління декількома типами токенів в одному контракті. Він дозволяє передавати і торгувати декількома типами токенів в одній транзакції, зменшуючи витрати на байт-код.

Стандарт ERC-6551 інноваційно використовує ідентифікацію та управління обліковими записами для NFT. Він дозволяє NFT функціонувати як повноцінні рахунки, зберігати активи і взаємодіяти з dApps, відкриваючи нові можливості для Web3-ігор та використання кількох гаманців у різних мережах.

Проаналізувавши використання Etherscan.io, пошукової системи на блокчейні Ethereum, для відстеження та перевірки транзакцій, контрактів та адрес, було встановлено, що хоча вона надає вичерпні дані, бракує інструментів для боротьби з фішинговими транзакціями. Фішингові атаки, які змушують користувачів переказувати кошти на фальшиві адреси, стали поширеними і загрожують безпеці багатьох адрес, тому рекомендується проявляти обережність та перевіряти контракти токенів.

РОЗДІЛ 3
ТЕХНОЛОГІЯ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУ ТА ПРИЙОМУ ДАНИХ

Отже потрібно розробити файлову структуру для запобігання фішинговим атакам в однорангових мережах, яка забезпечуватиме ефективне аналіз, виявлення та ліквідування підроблених елементів, ідентифікацію характерних ознак фішингу та забезпечуватиме безпеку користувачів під час он-чейн аналізу.

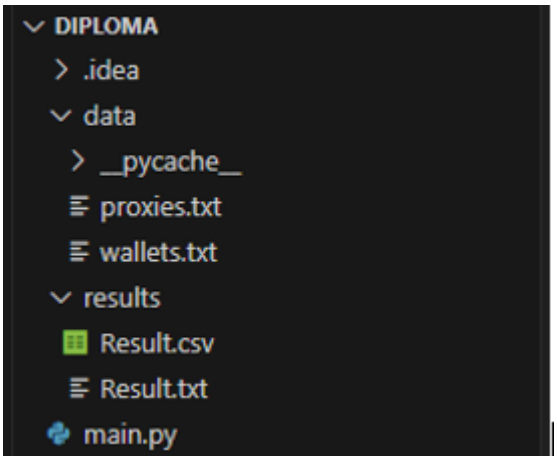


Рис. 3.1. Файлова структура проєкту

3.1. Розробка файлової структури

Структура файлу коду складається з наступних компонентів:

Папка Diploma: Ця папка містить файли даних, необхідні для роботи коду.

proxies.txt: Цей файл містить список проксі-серверів, які використовуються для виконання HTTP-запитів. Проксі зчитуються з цього файлу і вибираються випадковим чином під час запитів до API Debank.

wallets.txt: Цей файл містить список крос-чейн адресів . Ці адреси зчитуються з файлу і використовуються для отримання даних з API Debank для кожного адресу.

Папка Result: Ця папка використовується для зберігання вихідних результатів, згенерованих кодом.

Result.csv: Це файл у форматі CSV (Comma-Separated Values), який містить звіт про дані гаманця. Він містить такі колонки, як номер гаманця, адреса гаманця, баланс, інвестиції в протокол, інвестиції в токени і суми NFT.

Result.txt: Цей файл є текстовим файлом, який містить той самий підсумковий звіт, що і CSV-файл. Він забезпечує читабельний формат даних гаманця з правильним форматуванням і відступами.

main.py: Це основний файл коду, який містить Python-скрипт для отримання та аналізу даних з API Debank. Він включає необхідні імпортні дані, конфігураційні змінні, функції та основну логіку виконання.

Код побудовано за модульним принципом, з окремими файлами для зберігання даних, генерації результатів та основної логіки коду. Така структура файлів дозволяє легко організувати, підтримувати та розширювати код.

3.2. Етапи будування аналітичної програми:

Потрібно розробити архітектуру програмної системи , яка буде включати компоненти для ефективного виявлення, аналізу та блокування елементів, ідентифікації характерних ознак фішингу. Система повинна вміти

надсилати запити, збирати та обробляти дані, записувати результат у таблиці. Крім того, система повинна мати можливість швидкого реагування на виявлені загрози та ліквідувати їх.

- Аналіз: Розуміння вимог і завдань програми, включаючи отримання даних з API Debank для декількох адресів і створення зведеного звіту.
- Планування: Визначити необхідні інструменти та бібліотеки, необхідні для програми, такі як aiohttp для асинхронних HTTP-запитів та tabulate для форматування таблиць.
- Структура коду: Створення файлової структури з окремими папками для даних і результатів, а також файл main.py, що містить логіку коду.
- Обробка даних: Реалізація функцій для завантаження інформації про адреси гаманців та проксі з файлів даних, а також визначення структури для зберігання отриманих даних.
- Асинхронний пошук: Використання asyncio та aiohttp для реалізації паралельних та асинхронних API-запитів для отримання даних для кожного гаманця та модуля.
- Обробка даних: Обробка отриманих даних для вилучення відповідної інформації, такої як залишки токенів, NFT-колекції та протоколи для кожного гаманця.
- Створення звітів: Створення зведеного звіту з використанням оброблених даних, включаючи баланси токенів, колекції NFT, протоколи і загальну вартість адресу в доларовому еквіваленті.
- Обробка вихідних даних: Запис звіту як у текстовий файл, так і в CSV-файл для зручності аналізу великої кількості адресів та подальшої обробки.
- Обробка помилок: Реалізація реєстрації помилок та обробка винятків, щоб зафіксувати та обробити будь-які помилки, які можуть виникнути під час виконання програми.

- Тестування та доопрацювання: Тестування програму з різними гаманцями та сценаріями, необхідні доопрацювання та оптимізація на основі результатів тестування.
- Документація: Документування призначення, функціональність та використання програми, включаючи пояснення файлової структури, використаних інструментів та загального потоку виконання коду.

3.3. Середовище методів та технологій при будуванні з API Debank:

На етапі аналізу основною метою є глибоке розуміння вимог і завдань програми. Це передбачає визначення конкретних завдань, які повинна виконувати програма, включаючи отримання даних з API Debank для декількох гаманців і створення комплексного зведеного звіту.

Важливо ретельно вивчити документацію API Debank, щоб зрозуміти різні кінцеві запити і відповіді, які він пропонує, і структуру даних, які він надає. Це дозволить визначити конкретні API, які необхідно використовувати для ефективного отримання балансів токенів, колекцій NFT та протоколи.

Реалізація в коді:

```
# Debank
DEBANKS_API = {
    'token': 'https://api.debank.com/token/cache_balance_list?user_addr={}',
    'nft': 'https://api.debank.com/nft/collection_list?user_addr={}&chain={}',
    'protocol': 'https://api.debank.com/portfolio/project_list?user_addr={}',
}
```

Рис. 3.2. словник DEBANKS_API, який містить URL-адреси для різних типів запитів до Debank API.

```python
async def get_debank(session, address, type_, chain=''):
    while True:
        try:
            sleep = 3
            urls = {
                'token': f'https://api.debank.com/token/cache_balance_list?user_addr={address}',
                'nft': f'https://api.debank.com/nft/collection_list?user_addr={address}&chain={chain}',
                'protocol': f'https://api.debank.com/portfolio/project_list?user_addr={address}',
            }
            proxy = random.choice(PROXIES)
            async with session.get(urls[type_], proxy=proxy, timeout=10) as resp:
                if resp.status == 200:
                    resp_json = await resp.json(content_type=None)
                    if type_ == 'nft':
                        if resp_json['data']['job']:
                            await asyncio.sleep(sleep)
                        else:
                            get_result[type_][chain].update({address: resp_json})
                            logger.success(f'{address} | {type_} : {chain}')
                            break
                    else:
                        get_result[type_].update({address: resp_json})
                        logger.success(f'{address} | {type_}')
                        break
                else:
                    await asyncio.sleep(sleep)
        except Exception as error:
            logger.info(f'{address} | {type_} : {error}')
            await asyncio.sleep(3)
```

Рис. 3.3. Визначення асинхронної функції get_debank(), яка відповідає за виконання HTTP-запитів до Debank API.

Функція отримує декілька параметрів: session (сеанс aiohttp-клієнта), address (адреса гаманця для запиту), type_ (тип запиту, наприклад, 'token', 'nft' або 'protocol') та параметр chain.

Всередині циклу while функція створює відповідну URL-адресу на основі параметрів type_ і chain.

Вона також встановлює тривалість сну в 3 секунди.

Функція вибирає випадковий проксі зі списку PROXIES за допомогою random.choice(PROXIES).

Використовуючи async with, функція надсилає асинхронний GET-запит на сконструйовану URL-адресу, передаючи через обраний проксі і таймаут у 10 секунд. [22]

Список адресів:

Визначення отримання даних для декількох гаманців, що вимагає механізм завантаження і обробки списку адрес гаманців з файлу даних wallets.txt.

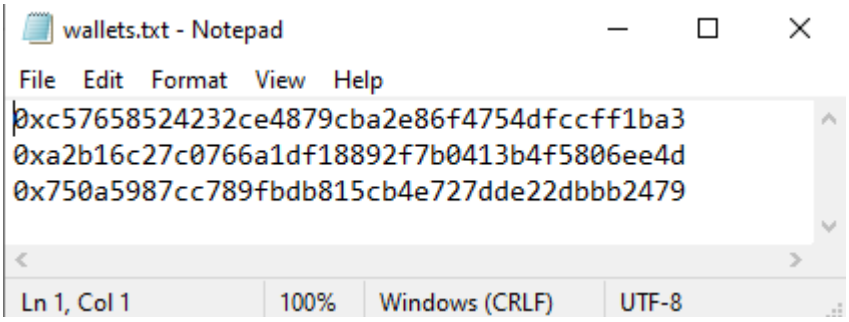


Рис. 3.4. Список адресів для аналізу.

```
# Configuration
WALLETS_FILE = "data/wallets.txt"
```

Рис. 3.5. Вказування шляху до текстового файлу з назвою "wallets.txt" в папці з назвою "data".

Цей рядок вказує на те, що програма очікує, що файл "wallets.txt" знаходиться в папці "data".

Передбачається, що файл містить адреси гаманців, причому кожна адреса перерахована з нового рядку.

```
28  # Load Wallets
29  with open(WALLETS_FILE, "r") as f:
30      WALLETS = [row.strip() for row in f]
31
```

Рис. 3.6. процес читання вмісту файлу "wallets.txt" та збереження адрес гаманців у списку з назвою WALLETS.

У рядках коду функція open() використовується для відкриття файлу, вказаного змінною WALLETS_FILE, у режимі читання. Об'єкт файлу присвоюється змінній f. Після цього використовується розуміння списку для ітерації по кожному рядку файлу і видалення всіх початкових і кінцевих пробілів, створюючи список адрес гаманців, що зберігається у змінній WALLETS.

Цей сегмент коду зчитує вміст вказаного файлу і витягує адреси гаманців для подальшого використання в програмі.

Proxies:

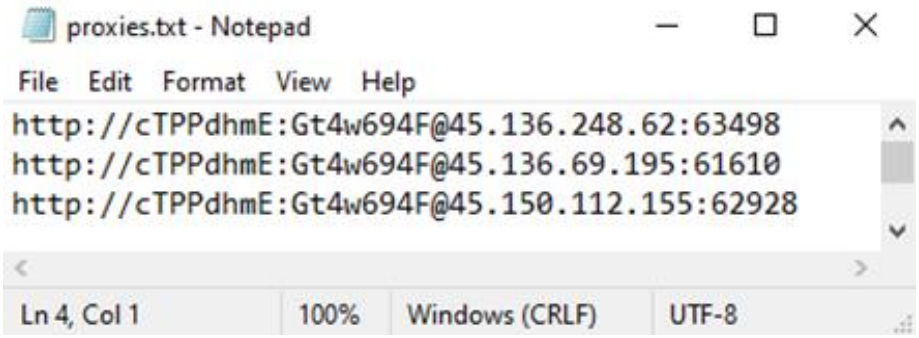


Рис. 3.7. Список проксі-серверів.

Файл proxies.txt містить список проксі-серверів, які можна використовувати для HTTP-запитів.

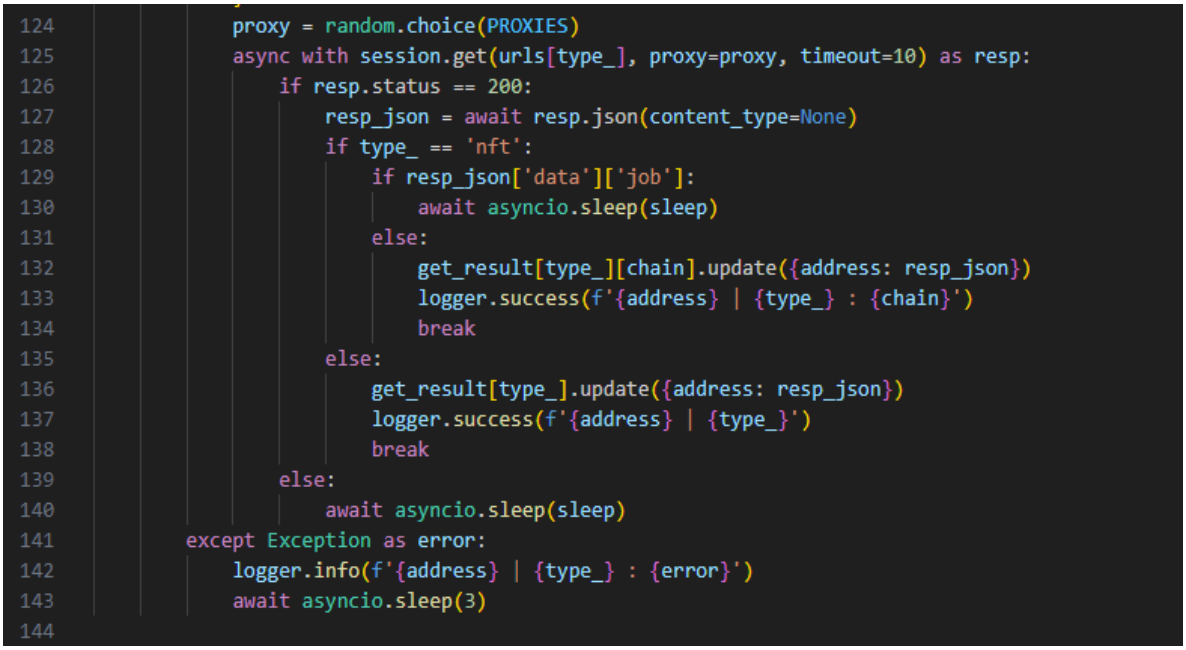
Проксі-сервери діють як посередники між клієнтом (кодом) і сервером, дозволяючи коду направляти свої запити через різні IP-адреси і порти.

```
15  # Configuration
16  WALLETS_FILE = "data/wallets.txt"
17  PROXIES_FILE = "data/proxies.txt"
```

Рис. 3.8. вказує шлях до текстового файлу з назвою "proxies.txt" в папці з назвою "data".

```
32  # Load Proxies
33  with open(PROXIES_FILE, "r") as f:
34      PROXIES = [row.strip() for row in f]
35
```

Рис. 3.9. процес читання вмісту файлу "proxies.txt" та збереження список проксі у списку з назвою PROXIES.



```
            proxy = random.choice(PROXIES)
            async with session.get(urls[type_], proxy=proxy, timeout=10) as resp:
                if resp.status == 200:
                    resp_json = await resp.json(content_type=None)
                    if type_ == 'nft':
                        if resp_json['data']['job']:
                            await asyncio.sleep(sleep)
                        else:
                            get_result[type_][chain].update({address: resp_json})
                            logger.success(f'{address} | {type_} : {chain}')
                            break
                    else:
                        get_result[type_].update({address: resp_json})
                        logger.success(f'{address} | {type_}')
                        break
                else:
                    await asyncio.sleep(sleep)
        except Exception as error:
            logger.info(f'{address} | {type_} : {error}')
            await asyncio.sleep(3)

```

Рис. 3.10. Цей фрагмент коду демонструє використання проксі-серверів для асинхронних HTTP-запитів за допомогою бібліотеки aiohttp.

3.4. Розробка функцій системи

Випадковий проксі вибирається зі списку проксі, що зберігається у змінній PROXIES. Потім за допомогою функції session.get() надсилається GET-запит на вказану URL-адресу з налаштуванням обраного проксі для виконання запиту.

Якщо код статусу відповіді дорівнює 200, дані відповіді обробляються і зберігаються в словнику get_result. Для запитів типу 'nft' виконується додаткова перевірка даних відповіді перед оновленням. Код фіксує успішність виконання запиту і виходить з циклу.

Якщо під час виконання запиту виникає виняток, він записується в журнал, і код чекає певний час, перш ніж повторити запит.

Цей механізм дозволяє коду робити HTTP-запити через різні проксі-сервери, забезпечуючи такі переваги, як балансування навантаження на Debank API та обхід обмежень за кількістю запитів.

Отримання даних:

Отримання даних у цьому коді відбувається переважно у функції get_debank() та пов'язаній з нею функції checker_main().

```
146 async def checker_main(modules, nft_chains, wallets):
147     async with aiohttp.ClientSession() as session:
148         tasks = []
149         for address in wallets:
150             if 'token' in modules:
151                 task = asyncio.create_task(get_debank(session, address, 'token'))
152                 tasks.append(task)
153             if 'protocol' in modules:
154                 task = asyncio.create_task(get_debank(session, address, 'protocol'))
155                 tasks.append(task)
156             if 'nft' in modules:
157                 for chain in nft_chains:
158                     task = asyncio.create_task(get_debank(session, address, 'nft', chain))
159                     tasks.append(task)
160         await asyncio.gather(*tasks)
161
```

Рис. 3.11. Асинхронна функція checker_main(), яка слугує центральною функцією для координації процесу отримання даних з різних джерел.

Функція checker_main() є критично важливим компонентом коду, що відповідає за ефективне отримання даних за допомогою асинхронного програмування. Вона використовує бібліотеку aiohttp для створення асинхронного клієнтського сеансу для виконання HTTP-запитів.

Всередині функції ініціалізується список завдань, який називається tasks, для зберігання асинхронних завдань. Він ітераційно проходить через наданий список адресів, перевіряючи наявність вказаних модулів (token, protocol і nft)для отримання даних.

Для кожного гаманця створюються завдання за допомогою asyncio.create_task() для модулів токена і протоколу. Потім ці завдання додаються до списку завдань.

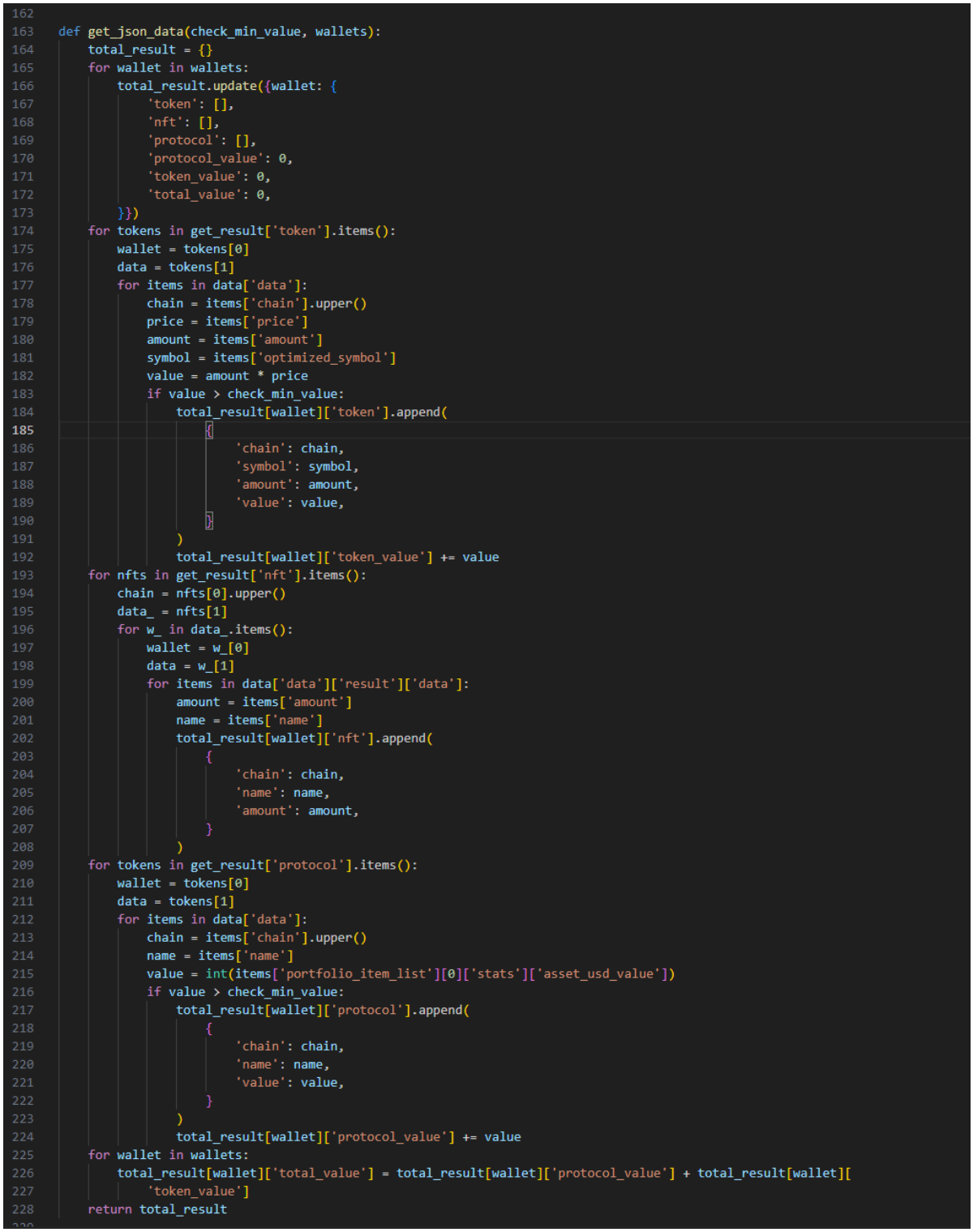
Функція також обробляє модуль 'nft' шляхом ітерації по списку nft_chains для отримання даних для кожного блокчейну. Завдання створюються для кожної комбінації гаманця, модуля і блокчейну та додаються до списку завдань.

asyncio.gather(*tasks) використовується для одночасного виконання всіх завдань. Такий підхід забезпечує паралельне виконання завдань пошуку даних, підвищуючи продуктивність і мінімізуючи час очікування.

Використовуючи асинхронне програмування, функція checker_main() дозволяє ефективно і паралельно отримувати дані з декількох джерел. Вона максимізує використання доступних ресурсів і значно скорочує загальний час виконання, що робить її добре придатною для обробки великомасштабних завдань пошуку даних.

Обробка даних:

Обробка даних у наведеному коді відбувається переважно у функціях get_json_data() та send_result().



```
def get_json_data(check_min_value, wallets):
    total_result = {}
    for wallet in wallets:
        total_result.update({wallet: {
            'token': [],
            'nft': [],
            'protocol': [],
            'protocol_value': 0,
            'token_value': 0,
            'total_value': 0,
        }})
    for tokens in get_result['token'].items():
        wallet = tokens[0]
        data = tokens[1]
        for items in data['data']:
            chain = items['chain'].upper()
            price = items['price']
            amount = items['amount']
            symbol = items['optimized_symbol']
            value = amount * price
            if value > check_min_value:
                total_result[wallet]['token'].append(
                    {
                        'chain': chain,
                        'symbol': symbol,
                        'amount': amount,
                        'value': value,
                    }
                )
                total_result[wallet]['token_value'] += value
    for nfts in get_result['nft'].items():
        chain = nfts[0].upper()
        data_ = nfts[1]
        for w_ in data_.items():
            wallet = w_[0]
            data = w_[1]
            for items in data['data']['result']['data']:
                amount = items['amount']
                name = items['name']
                total_result[wallet]['nft'].append(
                    {
                        'chain': chain,
                        'name': name,
                        'amount': amount,
                    }
                )
    for tokens in get_result['protocol'].items():
        wallet = tokens[0]
        data = tokens[1]
        for items in data['data']:
            chain = items['chain'].upper()
            name = items['name']
            value = int(items['portfolio_item_list'][0]['stats']['asset_usd_value'])
            if value > check_min_value:
                total_result[wallet]['protocol'].append(
                    {
                        'chain': chain,
                        'name': name,
                        'value': value,
                    }
                )
                total_result[wallet]['protocol_value'] += value
    for wallet in wallets:
        total_result[wallet]['total_value'] = total_result[wallet]['protocol_value'] + total_result[wallet][
            'token_value']
    return total_result
```

Рис. 3.12. Функція get_json_data(check_min_value, wallets) отримує в якості параметрів поріг мінімального значення в доларовому еквіваленті і список адресів.

Вона обробляє отримані дані, що зберігаються в словнику get_result. Функція ітераційно перебирає дані токена, nft і протоколу, виконуючи обчислення, фільтрацію на основі мінімального порогу значення і

організовуючи дані в структурований формат. Оброблені дані повертаються у вигляді словника, що містить вичерпну інформацію про залишки токенів, NFT і значення протоколів для кожного гаманця.

```python
def send_result(get_json, file_name, check_chain, check_coin):
    file = open(f'{outfile}results/{file_name}.txt', 'w', encoding='utf-8')
    with open(f'{outfile}results/{file_name}.csv', 'w', newline='') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quoting=csv.QUOTE_MINIMAL)
        spamwriter.writerow(['number', 'wallet', 'balance $', 'protocols $', 'tokens $', 'nft amount'])
        all_wallets_value = []
        all_finder_token = []
        zero = 0
        for wallets in get_json.items():
            zero += 1
            wallet = wallets[0]
            data = wallets[1]
            file.write(f'\n{zero}. {wallet}\n')
            tokens = []
            for items in data['token']:
                chain = items['chain']
                symbol = items['symbol']
                amount = round_to(items['amount'])
                value = int(items['value'])
                tokens.append([chain, amount, symbol, f'{value} $'])
                if check_chain != '':
                    if check_chain == chain:
                        if check_coin == symbol:
                            finder = amount
                else:
                    if check_coin == symbol:
                        finder = amount
            protocols = []
            for items in data['protocol']:
                chain = items['chain']
                name = items['name']
                value = int(items['value'])
                protocols.append([chain, name, f'{value} $'])
            nfts = []
            nft_amounts = []
            for items in data['nft']:
                chain = items['chain']
                name = items['name']
                amount = int(items['amount'])
                nft_amounts.append(amount)
                nfts.append([chain, name, amount])
```

```
            protocol_value = int(data['protocol_value'])
            token_value = int(data['token_value'])
            total_value = int(data['total_value'])
            table_type = 'double_outline'
            head_table = ['chain', 'amount', 'symbol', 'value']
            tokens_ = tabulate(tokens, head_table, tablefmt=table_type)
            head_table = ['chain', 'name', 'value']
            protocols_ = tabulate(protocols, head_table, tablefmt=table_type)
            head_table = ['chain', 'name', 'amount']
            nfts_ = tabulate(nfts, head_table, tablefmt=table_type)
            cprint(f'\n{zero}. {wallet}\n', 'yellow')
            if len(tokens) > 0:
                file.write(f'\n{tokens_}\n')
                cprint(tokens_, 'white')
            if len(protocols) > 0:
                file.write(f'\n{protocols_}\n')
                cprint(protocols_, 'white')
            if len(nfts) > 0:
                file.write(f'\n{nfts_}\n')
                cprint(nfts_, 'white')
            file.write(f'\ntotal_value : {total_value} $\n')
            cprint(f'\ntotal_value : {total_value} $\n', 'green')
            if check_coin != '':
                file.write(f'{check_coin} : {finder}\n')
                cprint(f'{check_coin} : {finder}\n', 'green')
                all_finder_token.append(finder)
            spamwriter.writerow([zero, wallet, total_value, protocol_value, token_value, sum(nft_amounts)])
            all_wallets_value.append(total_value)
        cprint(f'\n>>> ALL WALLETS VALUE : {sum(all_wallets_value)} $ <<<\n', 'blue')
        file.write(f'\n>>> ALL WALLETS VALUE : {sum(all_wallets_value)} $ <<<\n')
        spamwriter.writerow(['ALL_VALUE :', sum(all_wallets_value)])
        amount_finder_coin = round_to(sum(all_finder_token))
        if amount_finder_coin > 0:
            cprint(f'\n>>> {check_coin} : {amount_finder_coin}  <<<\n', 'blue')
            file.write(f'\n>>> {check_coin} : {amount_finder_coin}  <<<\n')
            spamwriter.writerow([f'{check_coin}', amount_finder_coin])
    file.close()
    cprint(f'results : {outfile}{file_name}.csv и {outfile}{file_name}.txt\n', 'blue')

```

Рис. 3.13. Функція send_result(get_json, file_name, check_chain, check_coin) отримує на вхід оброблені дані, ім'я файлу, перевіряє на валідність блокчейн і токен.

Функція send_result().займається, в основному, формуванням звіту. Ця функція отримує на вхід оброблені дані, ім'я файлу та додаткові параметри. Вона починає роботу з відкриття вихідних файлів для запису результатів - текстового та CSV-файлу. Функція виконує подальшу обробку оброблених даних для створення звіту. Вона обчислює загальну вартість гаманців, використовує бібліотеку таблиць для форматування таблиць балансів токенів, значень протоколів і NFT. Потім результати записуються у вихідні файли в читабельному текстовому форматі і форматі з розділенням комами, придатному для подальшого аналізу. Крім того, функція використовує функцію cprint для друку інформативних повідомлень на консоль, надаючи оновлення та ключові результати. Інкапсулюючи логіку генерації звітів, код

забезпечує організовану та ефективну обробку згенерованих звітів, дозволяючи користувачам ефективно аналізувати та інтерпретувати оброблені дані.

Разом ці дві функції обробляють отримані дані, виконують обчислення, фільтрацію та структурування даних, формують вихідні файли та звіти з узагальненням проаналізованої інформації. Вони забезпечують комплексний аналіз вартості гаманців, балансів токенів, колекцій NFT і значень протоколів, що дозволяє отримувати інформацію і приймати обґрунтовані рішення на основі оброблених даних.

Асинхронні функції async def get_debank() і async def checker_main():

Асинхронна функція get_debank() отримує дані з API Debank в асинхронний спосіб. Вона приймає такі параметри, як об'єкт сеансу, адреса гаманця, тип даних і необов'язковий блокчейн.

Використовуючи асинхронність з session.get(), вона надсилає асинхронний HTTP-запит, дозволяючи іншим завданням продовжувати роботу під час очікування відповіді.

Функція checker_main() слугує центральним координатором для отримання даних. Вона встановлює асинхронну клієнтську сесію і створює завдання за допомогою asyncio.create_task() для виконання get_debank() асинхронно для кожної комбінації гаманців, модулів і ланцюжків.

Завдяки використанню функції очікування await asyncio.gather() всі завдання виконуються паралельно, забезпечуючи ефективну паралельну обробку завдань пошуку даних.

await asyncio.gather(): Цей вираз використовується для паралельного виконання декількох об'єктів, що очікують на виконання (таких як підпрограми). Він отримує в якості аргументів список об'єктів, що очікують на виконання, і очікує на їхнє колективне завершення. Виконання продовжується, коли всі очікувані об'єкти завершилися або згенерували

виключення. Дозволяє ефективно паралельно виконувати завдання і зазвичай використовується, коли потрібно виконати декілька асинхронних операцій одночасно.

await asyncio.sleep(sleep): Цей вираз призупиняє виконання поточної підпрограми на вказаний час, заданий параметром sleep. По суті, він вводить паузу або затримку у виконанні підпрограми, не блокуючи при цьому цикл обробки подій. Зазвичай використовується для введення періоду очікування або для імітації операцій, що вимагають багато часу, без зайвого використання системних ресурсів

"В результаті, await asyncio.gather() дозволяє паралельно виконувати декілька підпрограм, а await asyncio.sleep() вводить паузу у виконанні підпрограми. Вони служать різним цілям в управлінні потоком виконання в асинхронному програмуванні.

Ці асинхронні функції підвищують продуктивність програми, дозволяючи одночасно ініціювати декілька операцій пошуку даних і обробляти їх паралельно. Вони використовують переваги асинхронного програмування для підвищення швидкості реагування та ефективності при виконанні великомасштабних завдань пошуку даних.

Обробка помилок:

Обробка помилок у наведеному коді реалізована за допомогою блоків try-except та модуля logger з бібліотеки loguru.

У коді використовуються блоки try-except для перехоплення та обробки виключних ситуацій, які можуть виникнути під час виконання певних ділянок коду.

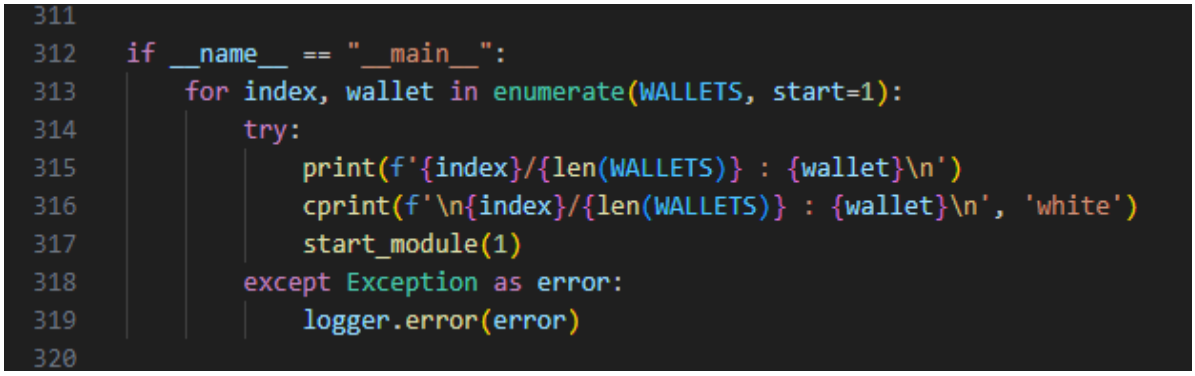


Рис. 3.14. Умова “if __name__ == "__main__":”.

У кількох словах, рядок “if __name__ == "__main__":” є умовою, яка перевіряє, чи запускається скрипт безпосередньо як основна програма, а не імпортується як модуль іншим скриптом.

В основному циклі коду використовується блок try-except для обробки будь-яких винятків, які можуть виникнути під час виконання циклу. Якщо виникає виняток, він перехоплюється блоком except, а повідомлення про помилку записується в журнал за допомогою методу logger.error(). Це дозволяє забезпечити належну обробку помилок і протоколювання будь-яких несподіваних помилок, які можуть виникнути під час виконання циклу.

Ці приклади демонструють, як блоки try-except використовуються в коді для перехоплення та обробки специфічних виключень, забезпечуючи механізм обробки помилок та витончену поведінку при виникненні помилок.

Модуль logger з бібліотеки loguru використовується для реєстрації повідомлень про помилки та винятки. Наприклад, у функції get_debank(), якщо під час HTTP-запиту до API Debank виникає виняток, повідомлення про помилку реєструється за допомогою методу loger.info(). Це допомагає в усуненні несправностей і виявленні причини помилок при отриманні даних.

Обробка помилок також передбачає використання блоків except для вказівки типу виключення, що перехоплюється. Вказавши конкретний клас винятків, код може обробляти різні типи помилок окремо і виконувати відповідні дії.

Механізм обробки помилок допомагає підтримувати стабільність програми, запобігаючи тому, щоб необроблені винятки не призводили до

завершення роботи програми або непередбачуваної поведінки. Він дозволяє коду витончено обробляти помилки, реєструвати відповідну інформацію та продовжувати виконання або надавати відповідний зворотній зв'язок користувачеві.

Загалом, обробка помилок є важливим аспектом коду, який гарантує, що потенційні винятки або помилки будуть перехоплені, зареєстровані та оброблені належним чином, сприяючи підвищенню надійності та стійкості програми.

3.5. Реалізація інструментів та бібліотек, потрібних для роботи системи

1. JSON

Бібліотека json - це стандартна бібліотека в Python, яка надає функціональність для кодування і декодування даних JSON. У коді вона використовується для запису отриманих даних з API Debank у JSON-файл.

2. Time

Модуль time в Python надає різні функції для роботи з операціями, пов'язаними з часом. У коді він використовується для введення затримок між запитами до API, щоб уникнути перевантаження сервера та забезпечити більш реалістичну картину використання.

3. Random

Бібліотека random в Python використовується для генерації випадкових значень. У коді вона використовується для вибору випадкового проксі зі списку проксі і для введення випадкових інтервалів сну між запитами API.

4. CSV

Модуль csv в Python надає функціональність для читання і запису файлів CSV. У коді він використовується для генерації CSV-звіту, що містить зведені дані про гаманці.

5. Math

Бібліотека math в Python надає різні математичні операції та функції. У коді вона використовується для округлення чисел до заданої кількості знаків після коми.

6. Asyncio

Бібліотека asyncio - це фреймворк для написання асинхронного коду з використанням підпрограм, циклів обробки подій і завдань. У коді вона використовується для реалізації асинхронних завдань для здійснення паралельних API-запитів до API Debank.

7. Aiohttp

Бібліотека aiohttp - це асинхронний HTTP-клієнт для Python. Вона дозволяє робити HTTP-запити асинхронно, що сприяє підвищенню продуктивності при одночасному виконанні декількох запитів до API.

8. Loguru

Бібліотека loguru - це бібліотека логування, яка надає більш зручний та виразний синтаксис для логування в Python. У коді вона використовується для реєстрації інформаційних повідомлень та повідомлень про помилки в процесі отримання даних.

9. Termcolor

Бібліотека termcolor надає функціонал для розфарбовування та форматування тексту, що виводиться в консолі. У коді вона використовується для виділення та диференціації певних повідомлень, що виводяться.

10. Tqdm

Бібліотека tqdm використовується для створення індикаторів виконання циклів і завдань. У коді вона використовується для відображення індикатора виконання під час очікування під час сплячих інтервалів.

11. Tabulate

Ця бібліотека використовується для форматування даних у структуровані таблиці. Вона допомагає створювати зручні для читання та організовані табличні представлення балансів токенів, значень протоколів та сум NFT. Вона покращує представлення даних у створених звітах.

3.6. Аналіз структури роботи аналітичної системи.

Аналіз структури роботи аналітичної системи:

− Програма починається з імпорту необхідних модулів та визначення конфігураційних змінних.

− Завантажує адреси гаманців з файлу wallets.txt і зберігає їх у списку WALLETS.

− Вона завантажує адреси проксі-серверів з файлу proxies.txt і зберігає їх у списку PROXIES.

− У програмі визначено кілька утиліт для перетворень, роботи з файлами і переходу в режим сну.

− Функція get_debank виконує асинхронні API-запити для отримання даних з API Debank на основі наданих параметрів.

− Функція checker_main координує асинхронне виконання API-запитів для декількох гаманців, модулів і ланцюжків.

− Функція get_json_data обробляє отримані дані і повертає словник (total_result), що містить релевантну інформацію.

− Функція send_result записує оброблені дані у файли Result.txt та Result.csv.

− Функція start_debank ініціалізує змінні, викликає інші функції та обробляє результати.

− Функція start_module викликається зі значенням модуля для запуску функції start_debank.

− Блок if __name__ == "__main__": виконує основний код. Він перебирає список WALLETS і викликає start_module(1) для кожного гаманця.

− Будь-які винятки, що виникають під час виконання коду, протоколюються як помилки.

Підготовка списків адресів у файлі wallets.txt.

Програма підтримує вибірку даних для декількох адрес гаманців. Кількість адрес, можна використовувати, залежить від конкретних потреб і обмежень API Debank. Код не накладає жодних жорстких обмежень на кількість обчисливальних адрес.

Щоб надати адреси програмі, потрібно створити текстовий файл з назвою wallets.txt у папці data/ Кожен рядок у файлі повинен містити одну адресу гаманця. Можна отримати ці адреси з різних джерел, таких як ваші власні гаманці, або зі сканерів як Debank I etherscan, bscscan і т.д.

Коли wallets.txt заповнений потрібними адресами гаманців. Програма буде зчитувати адреси з цього файлу і виконувати необхідні API-запити для кожної адреси.

При роботі з зовнішніми API важливо дотримуватися будь-яких обмежень швидкості або правил використання, встановлених постачальником API.

Ретельно керуючи кількістю адрес, дотримуючись обмежень API, можна ефективно використовувати програму для отримання даних за значною кількістю адрес гаманців.

Всього було обрано 3 адреси із самого сканеру Debank

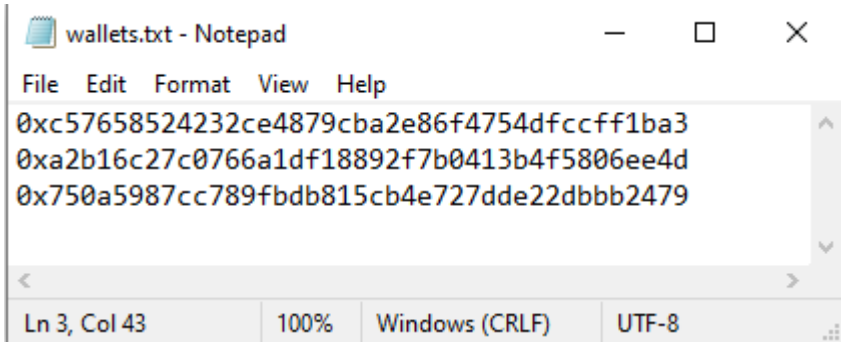


Рис. 3.15.Обрані адреси.

Включення в файл proxies.txt адреси проксі-серверів для запитів до Debank API.

API Debank має певні обмеження, про які при запиті ми повинні знати. Хоча конкретні обмеження можуть відрізнятися і можуть змінюватися, деякі загальні обмеження, з якими можемо зіткнутися, включають в себе наступні:

- Отримати загальний баланс користувача: 30
- Отримати баланс токенів користувача в ланцюзі: 6
- Отримати баланс токенів користувача у всіх підтримуваних блокчейнах: 18
- Отримати історію транзакцій користувача: 5
- Отримати історію транзакцій користувача у всіх підтримуваних блокчейнах: 15
- Отримати кешований баланс користувача в протоколі: 4
- Отримати кешований баланс користувача на всіх підтримуваних блокчейнах у протоколі: 12
- Отримати детальний кешований портфель користувача в ланцюзі в протоколі: 10
- Отримати кешовані портфелі користувачів на всіх підтримуваних блокчейнах у протоколі: 30
- Отримати список NFT користувачів у ланцюжку: 6
- Отримати список NFT користувачів на всіх підтримуваних ланцюжках: 18
- Отримати баланс по блокчейну: 10
- Отримати баланс конкретного токена: 1
- Отримати портфель користувача в реальному часі в протоколі: 4
- Отримати список ланцюжків, які використовує користувач: 2
- Отримати поточний список авторизації токенів користувача: 10
- Отримати поточний список авторизації NFT користувача: 10
- Отримати 24-годинну нетто-криву користувача по одному ланцюжку: 30
- Отримати 24-годинну нетто-криву користувача за всіма ланцюгами: 90

API Debank може встановлює обмеження на кількість запитів, які можемо зробити протягом певного періоду часу. Існує обмеження на кількість дозволених одночасних запитів. Це означає, що потрібно керувати своїми запитами і уникати перевантаження API занадто великою кількістю одночасних з'єднань. Асинхронні методи програмування, подібні до тих, що

використовуються в наданому коді, можуть допомогти керувати паралельними запитами більш ефективно.

Код програми включає в себе можливість використання проксі-серверів при виконанні запитів до API. Проксі-сервери діють як посередники між програмою і сервером API Debank.

Проксі-сервери дозволяють робити запити до Debank API з різних IP-адрес. Це може бути корисно в ситуаціях, коли ми тестуємо програму, де за певний час вона запускається декілька десятків разів або запитуємо інформацію по великій кількості адресів одночасно. Потрібно розподілити запити між кількома IP-адресами, підвищити загальну продуктивність і запобігти перевантаженню одного сервера або IP-адреси.

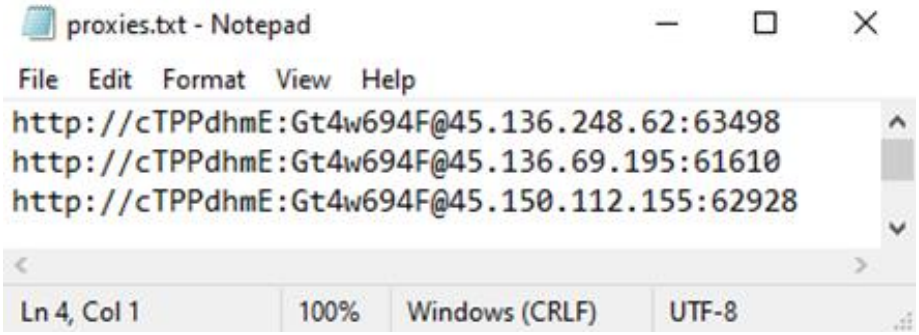


Рис. 3.16. Обрані проксі-сервери.

Файл proxies.txt в папці data/, як зазначено в структурі файлів, використовується для зберігання списку проксі-адрес. Програма випадковим чином вибирає проксі з цього списку для кожного запиту API, гарантуючи, що запити розподіляються між різними проксі.

Ефективно використовуючи проксі-сервери, ми покращуємо швидкість управління запитами, та зменшуємо обмеження, що накладаються API Debank.

3.7. Реалізація системи та вихідні дані

Реалізація системи та вихідні дані представлені на рис. 3.17. – 3.19.

```
1/3 : 0xc57658524232ce4879cba2e86f4754dfccff1ba3

2023-06-14 16:28:16.673 | SUCCESS  | __main__:get_debank:126 - 0x750a5987cc789fbdb815cb4e727dde22dbbb2479 | token
2023-06-14 16:28:16.712 | SUCCESS  | __main__:get_debank:126 - 0xc57658524232ce4879cba2e86f4754dfccff1ba3 | token
2023-06-14 16:28:16.783 | SUCCESS  | __main__:get_debank:126 - 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d | token
2023-06-14 16:28:16.797 | SUCCESS  | __main__:get_debank:126 - 0x750a5987cc789fbdb815cb4e727dde22dbbb2479 | protocol
2023-06-14 16:28:17.080 | SUCCESS  | __main__:get_debank:126 - 0xc57658524232ce4879cba2e86f4754dfccff1ba3 | protocol
2023-06-14 16:28:17.519 | SUCCESS  | __main__:get_debank:126 - 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d | protocol
```

Рис. 3.17. Виконання в терміналі 1-го адресу.

```
2/3 : 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d

2023-06-14 16:28:18.027 | SUCCESS  | __main__:get_debank:126 - 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d | token
2023-06-14 16:28:18.051 | SUCCESS  | __main__:get_debank:126 - 0x750a5987cc789fbdb815cb4e727dde22dbbb2479 | token
2023-06-14 16:28:18.156 | SUCCESS  | __main__:get_debank:126 - 0xc57658524232ce4879cba2e86f4754dfccff1ba3 | protocol
2023-06-14 16:28:18.716 | SUCCESS  | __main__:get_debank:126 - 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d | protocol
2023-06-14 16:28:24.877 | SUCCESS  | __main__:get_debank:126 - 0xc57658524232ce4879cba2e86f4754dfccff1ba3 | token
2023-06-14 16:28:24.989 | SUCCESS  | __main__:get_debank:126 - 0x750a5987cc789fbdb815cb4e727dde22dbbb2479 | protocol
```

Рис. 3.18. Виконання в терміналі 2-го адресу.

```
3/3 : 0x750a5987cc789fbdb815cb4e727dde22dbbb2479

2023-06-14 16:28:25.513 | SUCCESS  | __main__:get_debank:126 - 0x750a5987cc789fbdb815cb4e727dde22dbbb2479 | protocol
2023-06-14 16:28:25.683 | SUCCESS  | __main__:get_debank:126 - 0xc57658524232ce4879cba2e86f4754dfccff1ba3 | protocol
2023-06-14 16:28:25.875 | SUCCESS  | __main__:get_debank:126 - 0xc57658524232ce4879cba2e86f4754dfccff1ba3 | token
2023-06-14 16:28:28.974 | SUCCESS  | __main__:get_debank:126 - 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d | token
2023-06-14 16:28:29.172 | SUCCESS  | __main__:get_debank:126 - 0x750a5987cc789fbdb815cb4e727dde22dbbb2479 | token
2023-06-14 16:28:32.904 | SUCCESS  | __main__:get_debank:126 - 0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d | protocol
```

Рис. 3.19. Виконання в терміналі 3-го адресу.

Програма отримує дані з API Debank, обробляє їх і створює файли Result.txt і Result.csv у папці results/.

chain	amount	symbol	value
OP	17.314	OP	29 $
OP	0.115	ETH	209 $
ARB	0.0615	ETH	111 $
ARB	0.92	LINK	6 $
ARB	1.2	LPT	5 $
BSC	227.60	PSP	7 $
BSC	49.24	SD	44 $
BSC	0.0501	BNB	15 $
ERA	0.166	ETH	302 $
ERA	0.00475	WETH	8 $
ERA	32.755	USDC	32 $
ETH	52	MPS	393 $
ETH	0.155	ETH	281 $
FTM	30.803	FTM	11 $
PZE	0.795	ETH	1445 $
PZE	0.00418	WETH	7 $
PZE	280.072	QUICK	14 $
PZE	984.837	MATIC	860 $
PZE	55.904	DOV	9 $
AVAX	0.797	AVAX	11 $
XDAI	0.00537	WETH	9 $
MATIC	12.0921	USDT	12 $
MATIC	142.95	MATIC	124 $
MATIC	36259.8	JRT	361 $
MATIC	2831.91	PSP	93 $
MATIC	2243.61	MIMO	55 $
MATIC	5.163	LDO	11 $

)

chain	name	value
ARB	ArbDoge	31 $
ARB	Camelot	30 $
BSC	Alpaca Finance	6238 $
BSC	Beefy	3146 $
BSC	PancakeSwap V3	9 $
BSC	Stargate	84 $
ERA	iZUMi Finance	5216 $
ERA	Mute.io	27 $
ERA	SpaceFi	17 $
ERA	SyncSwap	18 $
ERA	Velocore	1004 $
FTM	Tarot	49 $
MATIC	0VIX	68850 $
MATIC	Aave V3	8 $
MATIC	Beefy	7775 $
MATIC	QiDao	9 $
MATIC	Metavault.Trade	537 $
MATIC	QuickSwap	16 $
MATIC	Stargate	22 $
OP	QiDao	1566 $
OP	Stargate	1098 $
ETH	ParaSwap	2583 $
PZE	0VIX	2570 $
PZE	QuickSwap	1688 $
ETH	SGN Celer Network	328 $

total_value : 107397 $
ETH : 0.0615

Рис. 3.20. Запис результату 1-го адресу в текстовому документі.

chain	amount	symbol	value
OP	296.422	USDT	296 $
OP	0.0281	ETH	51 $
ARB	470.056	USDT	470 $
ARB	69.502	USDC	69 $
ARB	0.0194	ETH	35 $
BSC	194.947	USDT	194 $
BSC	0.0758	BNB	23 $
ERA	0.0417	ETH	75 $
ETC	0.99	ETC	18 $
ETH	0.0657	ETH	119 $
ETH	27558.7	wCFG	6105 $
ETH	50663.7	PRIME	64080 $
ETH	20	BIT	10 $
ETH	12221.3	GFI	5242 $
PZE	0.0151	ETH	27 $
PZE	146.447	USDC	146 $
AVAX	1.464	AVAX	21 $
AVAX	248.611	USDt	248 $
KLAY	87.268	KLAY	15 $
NOVA	0.00276	ETH	5 $
EVMOS	73.0742	madUSDC	9 $
MATIC	100	STG	62 $
MATIC	249.627	USDT	249 $
MATIC	13.384	MATIC	11 $
METIS	6.239	m.USDC	6 $
AURORA	0.0312	BIFI	13 $

chain	name	value
ETH	Aave V2	24 $
ETH	Agility	9 $
ARB	Radiant Capital V2	83 $
AVAX	Stargate	1105 $
ERA	Nexon Finance	1709 $
ETH	Maple	6325 $
ETH	Merit Circle	6 $
ETH	Paragons DAO	6554 $

total_value : 93426 $
ETH : 0.0194

Рис. 3.21. Запис результату 2-го адресу в текстовому документі.

chain	amount	symbol	value
OP	0.0056	ETH	10 $
ARB	526.514	USDC	526 $
ARB	0.0432	ETH	78 $
ARB	20.505	DAI+	20 $
BSC	117.58	STG	73 $
BSC	0.0209	BNB	6 $
ERA	0.0338	ETH	61 $

chain	name	value
ARB	ArbDoge	6 $
ARB	GND Protocol	94 $
ARB	Stargate	19 $
AVAX	GMD Protocol	28630 $
AVAX	Stargate	1006 $
BSC	PancakeSwap	242 $
ETH	Lybra Finance	18 $
METIS	Tethys	45 $

total_value : 30837 $
ETH : 0.0432

Рис. 3.22. Запис результату 3-го адресу в текстовому документі і калькулятивний звіт основної нативної монети блокчейну.

number	wallet	balance $	protocols	tokens $
1	0xc57658524232ce4879cba2e86f4754dfccff1ba3	107219	102753	4466
2	0xa2b16c27c0766a1df18892f7b0413b4f5806ee4d	93194	15780	77414
3	0x750a5987cc789fbdb815cb4e727dde22dbbb2479	30839	30063	776
ALL_VALU	231252			
ETH	0.1241			

Рис. 3.23. Структурований звіт в .csv файлі.

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі представлено процес розробки системи, яка має змогу отримувати дані з API Debank для декількох адрес і генерувати результати, які можна перевірити і порівняти із зовнішніми джерелами. Завдяки ретельному виконанню та перевірці результатів, доказано функціональність системи.

Код був запущений з різними адресами гаманців, щоб отримати дані з API Debank. Було відстежено процес виконання на наявність помилок або винятків. Перевірено, що код успішно отримує дані для кожної адреси гаманця без будь-яких збоїв.

Було проаналізовано згенеровані файли результатів, включаючи Result.txt і Result.csv, розташовані в папці results/. Порівняно отримані залишки токенів, колекції NFT і списки протоколів/проектів з надійними зовнішніми джерелами. Отримані дані збігаються з даними, доступними на платформі Debank або Etherscan.io.

Проведено всебічний аналіз розрахованих загальних значень у результатах. Перевірено загальну вартість токенів, загальну вартість протоколів та загальну вартість гаманця за допомогою зовнішніх джерел. Розраховані значення відповідають очікуваним результатам і надають точне уявлення про стан гаманця.

В процесі тестування було оцінено продуктивність і масштабованість коду для більшої кількості адрес. Система ефективно обробляє збільшений обсяг запитів до API, зберігає точність і генерує результати, які збігаються із зовнішнім аналізом.

ВИСНОВКИ

Проведений огляд однорангових мереж та технологій блокчейну, який підтверджує їх широке використання у сучасному світі, оскільки вони дозволяють безпосередній обмін даними та спільне використання ресурсів між комп'ютерами. Однак, ці технології також мають вразливості, такі як атаки на вузли, поширення шкідливих файлів та фішингові атаки. Для забезпечення безпеки використовуються криптографічні протоколи, методи шифрування та механізми аутентифікації, а також додаткові заходи безпеки, такі як перевірка автентичності веб-сайтів та двофакторна автентифікація.

Огляд стандартів токенів в екосистемі Ethereum показує, що кожен стандарт (ERC-20, ERC-721, ERC-1155 та ERC-6551) виконує різні завдання і відповідає різноманітним сценаріям використання. Токени ERC-20 стали стандартом для створення та обміну взаємозамінних токенів, ERC-721 знайшли застосування в невзаємозамінних токенах (NFT), ERC-1155 пропонує управління декількома типами токенів, а ERC-6551 розширює можливості NFT.

Розроблена система, яка отримує дані з API Debank для декількох адрес, успішно функціонує і генерує результати, які можна перевірити і порівняти з зовнішніми джерелами. Проведений аналіз результатів підтверджує точність і надійність отриманих даних. Крім того, система ефективно працює з більшим обсягом запитів до API і зберігає точність при генерації результатів.

Отже, розроблена система підкреслює важливість безпеки в сучасному світі, зокрема у сфері однорангових мереж та розробки систем, пов'язаних з обробкою даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Підручник: Каран Сінгх Гарева, «Практичні блокчейни та криптовалюти», 2020. С. 137-148.

2. Наукова стаття: Махер Альхарбі, Аад ван Мурсел, «Смарт-контракти на основі блокчейну: Систематичне картографічне дослідження», 2017. С. 127-129.

3. Наукова стаття: ISDA(International Swaps and Derivatives Association), Лінклейтер,«Смарт-контракти та розподілений реєстр - юридична перспектива», 2017. С. 4-6.

4. Наукова стаття: Клак С.Д., Макгонагл К. «Розумні деривативні контракти: Генеральна угода ISDA та автоматизація платежів і поставок», 2019. С. 3-12.

5. Підручник: Гегедус П. «До аналізу ландшафту складності смарт-контрактів Ethereum на основі Solidity» У матеріалах 1-го Міжнародного семінару з нових тенденцій в інженерії програмного забезпечення для блокчейну, 2018. С. 35-39.

6. Ресурс Інтернету: «Розгортання смарт-контрактів на Ethereum підскочило на 293% у 2022 році: звіт розробників Alchemy» URL: https://decrypt.co/119371/ethereum-smart-contracts-deployment-jumped-293-2022-alchemy-developer-report (дата звернення 23.05.2023).

7. Ресур Інтернету: «Ethereum - про публічні та загальнодоступні блокчейни» URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains (дата звернення 24.05.2023).

8. Ресур Інтернету: Сила смарт-контрактів - автоматизація бізнес-процесів за допомогою блокчейну. URL: https://issuu.com/mavie-crypto/docs/the_power_of_smart_contracts (дата звернення 24.05.2023).

9. Ресур Інтернету: «СТАНДАРТ ТОКЕНІВ ERC-20» URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/ (дата звернення 24.05.2023).

10. Підручник: Тапскотт Д., Тапскотт А, «Революція блокчейну: Як технологія, що стоїть за біткойном, змінює гроші, бізнес та світ.» 2016. С. 14-19.

11. Ресур Інтернету: «Чому "DeFi" утопія - це фінанси без фінансистів: Швидкий перегляд». URL: https://www.bloomberg.com/news/articles/2020-08-26/why-defi-utopia-would-be-finance-without-financiers-quicktake (дата звернення 25.05.2023).

12. Ресур Інтернету: «Визначення NFT». URL: https://www.merriam-webster.com/dictionary/NFT (дата звернення 25.05.2023).

13. Ресур Інтернету: «СТАНДАРТ EIP-721». URL: https://eips.ethereum.org/EIPS/eip-721 (дата звернення 26.05.2023).

14. Ресур Інтернету: «СТАНДАРТ EIP-1155». URL: https://eips.ethereum.org/EIPS/eip-1155 (дата звернення 26.05.2023).

15. Ресур Інтернету: URL: «СТАНДАРТ EIP-6551». URL: https://eips.ethereum.org/EIPS/eip-6551 (дата звернення 26.05.2023).

16. Ресур Інтернету: «Атака з отруєнням - постійна загроза». URL: https://mirror.xyz/x-explore.eth/cL3d_CyNujXq8XY7ueP4omNXx_IY1EG5Dz0FD0vJ90M (дата звернення 26.05.2023).

17. Ресур Інтернету: «Тенденції щодо обсягів атак на bsc-токени». https://dune.com/queries/1675026/2772963 (дата звернення 29.05.2023).

18. Ресур Інтернету: «Тенденції щодо обсягів атак на eth-токени». https://dune.com/queries/1683373/2784897 (дата звернення 29.05.2023).

19. Ресур Інтернету: «Дані атаки про побудову першої та останньої адреси». URL: https://dune.com/opang/first-and-last-address-construction (дата звернення 28.05.2023).

20. Ресур Інтернету: «Загальна кількість жертв». URL: https://dune.com/queries/2320328/3797628 (дата звернення 29.05.2023).

21. Ресур Інтернету: «Викрадені кошти». URL: https://dune.com/queries/2325825/3807243 (дата звернення 29.05.2023).

22. Ресур Інтернету: «Ланцюжок». URL: https://docs.cloud.debank.com/en/readme/api-pro-reference/chain (дата звернення 01.06.2023).

ДОДАТОК А
Програмна реалізація коду:

```
import json
import time
import random
import csv
import math
import asyncio
import aiohttp
from loguru import logger
from termcolor import cprint
from tqdm import tqdm
from tabulate import tabulate


# Configuration
WALLETS_FILE = "data/wallets.txt"
PROXIES_FILE = "data/proxies.txt"


# Debank
DEBANKS_API = {
    'token': 'https://api.debank.com/token/cache_balance_list?user_addr={}',
    'nft': 'https://api.debank.com/nft/collection_list?user_addr={}&chain={}',
    'protocol': 'https://api.debank.com/portfolio/project_list?user_addr={}',
}
outfile = ''
# Load Wallets
with open(WALLETS_FILE, "r") as f:
    WALLETS = [row.strip() for row in f]
# Load Proxies
with open(PROXIES_FILE, "r") as f:
    PROXIES = [row.strip() for row in f]
get_result = {
    'token': {},
    'nft': {
        'op': {},
```

```
        'eth': {},
        'arb': {},
        'matic': {},
        'bsc': {},
    },
    'protocol': {}
}
def start_debank():
    wallets = WALLETS
    file_name, check_min_value, check_chain, check_coin, modules, nft_chains = value_debank()
    asyncio.run(checker_main(modules, nft_chains, wallets))
    get_json = get_json_data(check_min_value, wallets)
    send_result(get_json, file_name, check_chain, check_coin)
def start_module(module):
    if module == 1:
        start_debank()
def value_debank():
    modules = [
        'token',
        # 'nft',
        # 'nft',
        'protocol'
    ]
    nft_chains = [
        # 'op',
        'eth',
        # 'arb',
        # 'matic',
        # 'bsc'
    ]
    check_min_value = 5
    check_chain = 'ETH'
    check_coin = 'ETH'
    file_name = 'Result'
    return file_name, check_min_value, check_chain, check_coin, modules, nft_chains
def intToDecimal(qty, decimal):
```

```
    return int(qty * int("".join(["1"] + ["0"] * decimal)))
def decimalToInt(qty, decimal):
    return qty / int("".join((["1"] + ["0"] * decimal)))
def call_json(result, outfile):
    with open(f"{outfile}.json", "w") as file:
        json.dump(result, file, indent=4, ensure_ascii=False)
def sleeping(from_sleep, to_sleep):
    x = random.randint(from_sleep, to_sleep)
    for i in tqdm(range(x), desc='sleep ', bar_format='{desc}: {n_fmt}/{total_fmt}'):
        time.sleep(1)
async def get_debank(session, address, type_, chain=''):
    while True:
        try:
            sleep = 3
            urls = {
                'token': f'https://api.debank.com/token/cache_balance_list?user_addr={address}',
                'nft': f'https://api.debank.com/nft/collection_list?user_addr={address}&chain={chain}',
                'protocol': f'https://api.debank.com/portfolio/project_list?user_addr={address}',
            }
            proxy = random.choice(PROXIES)
            async with session.get(urls[type_], proxy=proxy, timeout=10) as resp:
                if resp.status == 200:
                    resp_json = await resp.json(content_type=None)
                    if type_ == 'nft':
                        if resp_json['data']['job']:
                            await asyncio.sleep(sleep)
                        else:
                            get_result[type_][chain].update({address: resp_json})
                            logger.success(f'{address} | {type_} : {chain}')
                            break
                    else:
                        get_result[type_].update({address: resp_json})
                        logger.success(f'{address} | {type_}')
                        break
                else:
                    await asyncio.sleep(sleep)
```

```
        except Exception as error:
            logger.info(f'{address} | {type_} : {error}')
            await asyncio.sleep(3)
async def checker_main(modules, nft_chains, wallets):
    async with aiohttp.ClientSession() as session:
        tasks = []
        for address in wallets:
            if 'token' in modules:
                task = asyncio.create_task(get_debank(session, address, 'token'))
                tasks.append(task)
            if 'protocol' in modules:
                task = asyncio.create_task(get_debank(session, address, 'protocol'))
                tasks.append(task)
            if 'nft' in modules:
                for chain in nft_chains:
                    task = asyncio.create_task(get_debank(session, address, 'nft', chain))
                    tasks.append(task)
        await asyncio.gather(*tasks)
def get_json_data(check_min_value, wallets):
    total_result = {}
    for wallet in wallets:
        total_result.update({wallet: {
            'token': [],
            'nft': [],
            'protocol': [],
            'protocol_value': 0,
            'token_value': 0,
            'total_value': 0,
        }})
    for tokens in get_result['token'].items():
        wallet = tokens[0]
        data = tokens[1]
        for items in data['data']:
            chain = items['chain'].upper()
            price = items['price']
            amount = items['amount']
```

```
            symbol = items['optimized_symbol']
            value = amount * price
            if value > check_min_value:
                total_result[wallet]['token'].append(
                    {
                        'chain': chain,
                        'symbol': symbol,
                        'amount': amount,
                        'value': value,
                    }
                )
                total_result[wallet]['token_value'] += value
    for nfts in get_result['nft'].items():
        chain = nfts[0].upper()
        data_ = nfts[1]
        for w_ in data_.items():
            wallet = w_[0]
            data = w_[1]
            for items in data['data']['result']['data']:
                amount = items['amount']
                name = items['name']
                total_result[wallet]['nft'].append(
                    {
                        'chain': chain,
                        'name': name,
                        'amount': amount,
                    }
                )
    for tokens in get_result['protocol'].items():
        wallet = tokens[0]
        data = tokens[1]
        for items in data['data']:
            chain = items['chain'].upper()
            name = items['name']
            value = int(items['portfolio_item_list'][0]['stats']['asset_usd_value'])
            if value > check_min_value:
```

```
                total_result[wallet]['protocol'].append(
                    {
                        'chain': chain,
                        'name': name,
                        'value': value,
                    }
                )
                total_result[wallet]['protocol_value'] += value
    for wallet in wallets:
        total_result[wallet]['total_value'] = total_result[wallet]['protocol_value'] + total_result[wallet][
            'token_value']
    return total_result
def round_to(num, digits=3):
    try:
        if num == 0: return 0
        scale = int(-math.floor(math.log10(abs(num - int(num))))) + digits - 1
        if scale < digits: scale = digits
        return round(num, scale)
    except:
        return num
def send_result(get_json, file_name, check_chain, check_coin):
    file = open(f'{outfile}results/{file_name}.txt', 'w', encoding='utf-8')
    with open(f'{outfile}results/{file_name}.csv', 'w', newline='') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quoting=csv.QUOTE_MINIMAL)
        spamwriter.writerow(['number', 'wallet', 'balance $', 'protocols $', 'tokens $', 'nft amount'])
        all_wallets_value = []
        all_finder_token = []
        zero = 0
        for wallets in get_json.items():
            zero += 1
            wallet = wallets[0]
            data = wallets[1]
            file.write(f'\n{zero}. {wallet}\n')
            tokens = []
            for items in data['token']:
                chain = items['chain']
```

```
        symbol = items['symbol']
        amount = round_to(items['amount'])
        value = int(items['value'])
        tokens.append([chain, amount, symbol, f'{value} $'])
        if check_chain != '':
            if check_chain == chain:
                if check_coin == symbol:
                    finder = amount
        else:
            if check_coin == symbol:
                finder = amount
    protocols = []
    for items in data['protocol']:
        chain = items['chain']
        name = items['name']
        value = int(items['value'])
        protocols.append([chain, name, f'{value} $'])
    nfts = []
    nft_amounts = []
    for items in data['nft']:
        chain = items['chain']
        name = items['name']
        amount = int(items['amount'])
        nft_amounts.append(amount)
        nfts.append([chain, name, amount])
    protocol_value = int(data['protocol_value'])
    token_value = int(data['token_value'])
    total_value = int(data['total_value'])
    table_type = 'double_outline'
    head_table = ['chain', 'amount', 'symbol', 'value']
    tokens_ = tabulate(tokens, head_table, tablefmt=table_type)
    head_table = ['chain', 'name', 'value']
    protocols_ = tabulate(protocols, head_table, tablefmt=table_type)
    head_table = ['chain', 'name', 'amount']
    nfts_ = tabulate(nfts, head_table, tablefmt=table_type)
    cprint(f'\n{zero}. {wallet}\n', 'yellow')
```

```
            if len(tokens) > 0:
                file.write(f'\n{tokens_}\n')
                cprint(tokens_, 'white')
            if len(protocols) > 0:
                file.write(f'\n{protocols_}\n')
                cprint(protocols_, 'white')
            if len(nfts) > 0:
                file.write(f'\n{nfts_}\n')
                cprint(nfts_, 'white')
            file.write(f'\ntotal_value : {total_value} $\n')
            cprint(f'\ntotal_value : {total_value} $\n', 'green')
            if check_coin != '':
                file.write(f'{check_coin} : {finder}\n')
                cprint(f'{check_coin} : {finder}\n', 'green')
                all_finder_token.append(finder)
            spamwriter.writerow([zero, wallet, total_value, protocol_value, token_value, sum(nft_amounts)])
            all_wallets_value.append(total_value)
        cprint(f'\n>>> ALL WALLETS VALUE : {sum(all_wallets_value)} $ <<<\n', 'blue')
        file.write(f'\n>>> ALL WALLETS VALUE : {sum(all_wallets_value)} $ <<<\n')
        spamwriter.writerow(['ALL_VALUE :', sum(all_wallets_value)])
        amount_finder_coin = round_to(sum(all_finder_token))
        if amount_finder_coin > 0:
            cprint(f'\n>>> {check_coin} : {amount_finder_coin}  <<<\n', 'blue')
            file.write(f'\n>>> {check_coin} : {amount_finder_coin}  <<<\n')
            spamwriter.writerow([f'{check_coin}', amount_finder_coin])
    file.close()
    cprint(f'results : {outfile}{file_name}.csv и {outfile}{file_name}.txt\n', 'blue')
if __name__ == "__main__":
    for index, wallet in enumerate(WALLETS, start=1):
        try:
            print(f'{index}/{len(WALLETS)} : {wallet}\n')
            cprint(f'\n{index}/{len(WALLETS)} : {wallet}\n', 'white')
            start_module(1)
        except Exception as error:
            logger.error(error)
RETRY = 0
```