МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ЧЕРКАСЬКИЙ
ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

МОНІТОРИНГ ВИТОКУ ОБЛІКОВИХ ДАНИХ У ВІДКРИТИХ ДЖЕРЕЛАХ

Виконав:
студент групи 2К-22 Спеціальності
123 «Комп’ютерна інженерія»
Іван ГЕТЬМАН
Керівник:
Маргарита МЕДОЛИЗ

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп’ютерної інженерії та інформаційних технологій

Спеціальність	123 «Комп’ютерна інженерія»
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувачка циклової комісії КІ та ІТ

_____Наталя ФАЛЬЧЕНКО

(підпис)

«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гетьману Івану Івановичу

1. Тема кваліфікаційної роботи «Моніторинг витоку облікових даних у відкритих джерелах»

   Керівник роботи Медолиз Маргарита Миколаївна, викладач вищої категорії, затверджені наказом закладу передвищої освіти від «06» жовтня 2025 року № 84/1у
2. Строк подання студентом кваліфікаційної роботи 08.06.2026
3. Вихідні дані до кваліфікаційної роботи: джерела з мережевої безпеки, наукові статті та електронні ресурси про алгоритми та методи виявлення витоків даних, платформи моделювання
4. Зміст кваліфікаційної роботи: аналіз витоку облікових даних; основні мережеві джерела поширення скомпрометованої інформації; розробка архітектури та внутрішньої структури системи моніторингу
5. Дата видачі завдання: 15.10.2025 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	Розділ 1 Аналіз загроз скомпрометованості облікових даних	09.12.2025	
3	Розділ 2 Проєктування системи моніторингу витоку облікових даних	10.03.2026	
4	Розділ 3 Практична реалізація та тестування системи моніторингу витоку облікових даних	28.04.2026	
5	Висновки	12.05.2026	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	05.06.2026	
8	Подання кваліфікаційної роботи на затвердження завідувачу ЦК (за 7 днів до захисту)	08.06.2026	

Студент _____ Іван ГЕТЬМАН
(підпис)

Керівник роботи _____ Маргарита МЕДОЛИЗ
(підпис)

АНОТАЦІЯ

У кваліфікаційній роботі було розроблено та реалізовано автоматизовану систему для моніторингу витоків облікових даних. Для її створення обрано сучасний асинхронний набір технологій на основі мови програмування Python і фреймворку aiogram, що дозволяє швидко та надійно обробляти велику кількість запитів. Система побудована за модульним принципом і складається з кількох взаємопов'язаних компонентів: модуля перевірки коректності вхідних ідентифікаторів, підсистеми взаємодії із зовнішніми сервісами кібербезпеки та аналітичного блоку для формування звітів про виявлені інциденти. Для перевірки працездатності розробленого рішення застосовано імітаційне моделювання, що дало змогу провести повноцінне тестування без використання комерційних ресурсів сторонніх сервісів. Отримані результати підтвердили ефективність запропонованої системи як інструмента для своєчасного виявлення загроз і підвищення рівня захисту персональних даних користувачів.

Ключові слова: OSINT, КІБЕРБЕЗПЕКА, ВИТІК ДАНИХ, PYTHON, AIOGRAM, АСИНХРОННА АРХІТЕКТУРА, АВТОМАТИЗОВАНИЙ МОНІТОРИНГ.

ABSTRACT

In the qualification work, an automated system for monitoring credential leaks was developed and implemented. To create it, a modern asynchronous set of technologies based on the Python programming language and the aiogram framework was chosen, which allows for fast and reliable processing of a large number of requests. The system is built on a modular principle and consists of several interconnected components: a module for checking the correctness of incoming identifiers, a subsystem for interaction with external cybersecurity services, and an analytical unit for generating reports on detected incidents. To verify the operability of the developed solution, simulation modeling was used, which made it possible to conduct full-fledged testing without using commercial resources of third-party services. The results obtained confirmed the effectiveness of the proposed system as a tool for timely detection of threats and increasing the level of protection of users' personal data.

Keywords: OSINT, CYBERSECURITY, DATA LEAKAGE, PYTHON, AIOGRAM, ASYNCHRONOUS ARCHITECTURE, AUTOMATED MONITORING.

ЗМІСТ

ВСТУП

У сучасному цифровому світі комп’ютерні мережі підтримують роботу ледь не кожної сфери життя. Соціальні платформи, електронна комерція, державні сервіси та банки щоденно використовують автентифікацію. Вона переважно спирається на базові облікові дані. Це логіни, адреси електронної пошти та паролі, які виконують роль ключів до інформаційних систем і визначають рівень прав доступу. Сьогодні кількість сервісів стрімко зростає. Відповідно, такі ідентифікатори набувають надзвичайної цінності, тому часто стають основною ціллю для хакерів та зловмисників. Втрата контролю над цими даними гарантовано веде до несанкціонованого доступу [2, 10]. Це означає крадіжку конфіденційних баз, відчутні фінансові втрати та інші критичні наслідки для бізнесу чи звичайних користувачів.

При цьому витоки інформації часто трапляються не лише через складні атаки, а через банальні помилки. Вразливості в коді, неправильне налаштування серверів адміністраторами або просто людський фактор [8]. Але найбільшу небезпеку становить те, що відбувається після крадіжки. Скомпрометовані дані масово зливаються у відкритий доступ. Вони опиняються на тіньових форумах, публікуються у вигляді гігантських текстових архівів та поширюються через спеціалізовані хакерські платформи. Зважаючи на таку доступність інформації, зловмисникам стає значно простіше реалізовувати автоматизовані атаки. Масовий перебір паролів чи захоплення чужих акаунтів стає питанням кількох хвилин. Традиційні методи захисту мережевого периметра тут безсилі. Вони просто не розраховані на ситуації, коли пароль вже вкрадено і він знаходиться поза контролем справжнього власника системи.

Саме тому в сучасних умовах критично необхідні системи проактивного моніторингу. Такі, що здатні безперервно парсити відкриті джерела, фіксувати факт витоку та негайно попереджати користувача. У сучасних реаліях найбільш швидким та безпечним способом взаємодії є використання ботів у популярних месенджерах [11, 13]. Ця архітектура дозволяє отримувати аналітику миттєво.

Клієнту не треба встановлювати підозрілі програми чи заходити на сумнівні сайти [6]. Усе відбувається в знайомому інтерфейсі, що значно підвищує довіру до інструменту. Відповідно, актуальність цього дослідження прямо пов'язана з необхідністю швидко виявляти вкрадені облікові дані та захищати користувачів за допомогою зручних діалогових систем [7].

Головна мета кваліфікаційної роботи полягає у розробці спеціального програмного засобу для моніторингу витоків у відкритих джерелах. Він має працювати на базі асинхронної взаємодії через інтерфейс месенджера, щоб забезпечити швидкий контроль та знизити ризик використання скомпрометованих паролів.

Для досягнення цієї мети необхідно виконати такі завдання:

1. Провести аналіз проблеми витоку облікових даних та оцінити ризики їх компрометації у сучасних мережах.
2. Дослідити ключові джерела, де найчастіше поширюється вкрадена інформація, та виконати огляд існуючих систем моніторингу .
3. Сформувати функціональні та безпекові вимоги до майбутнього програмного продукту.
4. Розробити концептуальну архітектуру та модульну структуру системи, інтегрувавши її з платформою месенджера .
5. Описати інженерну логіку роботи бота, скласти алгоритми і запропонувати безпечні методи взаємодії із зовнішніми REST API.

Об’єктом дослідження є процеси виникнення, подальшого поширення та інженерного виявлення витоків облікових даних у відкритих інформаційних джерелах сучасних комп'ютерних мереж.

Предметом дослідження є методи системного аналізу, архітектурні моделі та автоматизовані програмні засоби моніторингу витоку облікових даних у відкритих джерелах, реалізовані за допомогою діалогових інтерфейсів взаємодії.

Практичне значення отриманих - це готовий працездатний прототип асинхронного інструменту для пошуку злитих даних. Впровадження цієї розробки дозволить значно посилити захист особистих акаунтів і звести до

мінімуму успішність таргетованих атак. Завдяки високій гнучкості та мінімальним вимогам до "заліза", цей інструмент можна легко підключити до вже існуючих приватних або корпоративних систем кібербезпеки.

РОЗДІЛ 1 АНАЛІЗ ЗАГРОЗ СКОМПРОМЕТОВАНОСТІ ОБЛІКОВИХ ДАНИХ

Першочерговим завданням дослідження є концептуалізація поняття «облікові дані» та встановлення їхнього значення для безпеки цифрового простору. Доведено, що в сучасних умовах вони є фундаментальними ідентифікаторами доступу, які забезпечують санкціоноване функціонування як локальних корпоративних вебресурсів, так і розгалужених хмарних архітектур.

Також необхідно розглянути ключові фактори компрометації. Особливу увагу слід приділити аналізу механізмів компрометації приватних ідентифікаторів та чинників, що зумовлюють їх потрапляння у відкритий доступ.

1.1 Поняття облікових даних та їх роль в інформаційних системах

Облікові дані сьогодні виступають абсолютно базовим елементом архітектури будь-якої комп'ютерної системи. Саме вони відповідають за практичну реалізацію політик безпеки через три невід'ємні механізми: ідентифікацію, автентифікацію та авторизацію [9]. Якщо поглянути на це з боку комп'ютерної інженерії, то такі дані є просто унікальним набором цифрових маркерів, які дозволяють системі безпомилково зрозуміти, який саме суб'єкт намагається отримати доступ до ресурсів. Після перевірки справжності користувача система вирішує, які права йому делегувати. Взаємодія з апаратним забезпеченням, читання баз даних чи запуск програмного коду жорстко контролюється завдяки правильній автентифікації. Історично до цього масиву інформації входили виключно текстові логіни, адреси електронної пошти та класичні паролі.

Проте сучасна цифровізація кардинально змінила правила. Перелік ідентифікаторів суттєво розширився, і тепер до нього обов'язково входять сесійні ключі, складні криптографічні токени та динамічні коди для багатофакторної перевірки. Банківські платформи, державні електронні реєстри та закриті корпоративні мережі зараз повністю працюють на цих алгоритмах [2].

Окремо варто згадати про технології єдиного входу Single Sign-On. Сучасні мережі масово і дуже активно впроваджують таку централізовану автентифікацію, оскільки ця ідея максимально зручна для бізнесу. Один глобальний обліковий запис дає людині доступ одразу до десятків різних сервісів. Для адміністрування це величезний плюс: операційна ефективність зростає, навантаження на сервери падає, а користувачам не треба запам'ятовувати купу паролів. Але існує критична проблема, бо подібна централізація створює єдину точку відмови в усій архітектурі безпеки підприємства. Варто зловмиснику зламати лише один базовий ідентифікатор – і він автоматично отримує повний спектр прав на всі пов'язані ресурси [9]. На практиці ця ситуація додатково ускладнюється фактором людської ліні. Користувачі масово та свідомо ігнорують правила цифрової гігієни. Вони регулярно використовують абсолютно однакові або мінімально змінені паролі на різних, логічно не пов'язаних між собою сайтах. Успішний витік бази даних з якогось погано захищеного публічного форуму миттєво ставить під загрозу корпоративні пошти та фінансові акаунти цих же людей [7].

Хакери просто проганяють злиті бази через скрипти. Всі ці фактори у своїй сукупності створюють жорстку інженерну потребу розробляти спеціалізовані системи моніторингу [13]. Нам необхідні інструменти, здатні сканувати інформаційний простір та виявляти злиті дані ще до того, як по них вдарять автоматизовані кібератаки.

1.2 Причини витоку облікових даних

Інциденти інформаційної безпеки ніколи не стаються через одну дрібну помилку. Завжди спрацьовує цілий ланцюг обставин. Витоки формуються на стику технічних недопрацювань, поганої організації процесів та звичайного людського фактора. Щоб побудувати надійну систему моніторингу, треба розуміти саму природу цих явищ. Якщо подивитися на технічну сторону, то першопричиною найчастіше стають архітектурні дефекти програмного забезпечення. Зловмисники просто експлуатують вразливості, які розробники не

встигли або забули закрити. У хід ідуть ін'єкції баз даних та міжсайтовий скриптинг. Хакери б'ють прямо по механізмах автентифікації [8]. Особливо критичною ситуація стає тоді, коли конфіденційні записи лежать на серверах у відкритому вигляді або захищені старими алгоритмами шифрування. У таких випадках масовий злив даних – лише питання часу.

Окрема проблема полягає в мережевих атаках на етапі передачі інформації. Використання морально застарілих стеків протоколів без інтегрованого наскрізного кодування криптографічними методами дає змогу стороннім особам здійснювати прихований моніторинг і компрометацію облікових записів користувачів у процесі транзиту даних. Зловмисникам навіть не потрібно зламувати саму систему захисту схема такого перехоплення трафіку наведена в рисунку 1.1

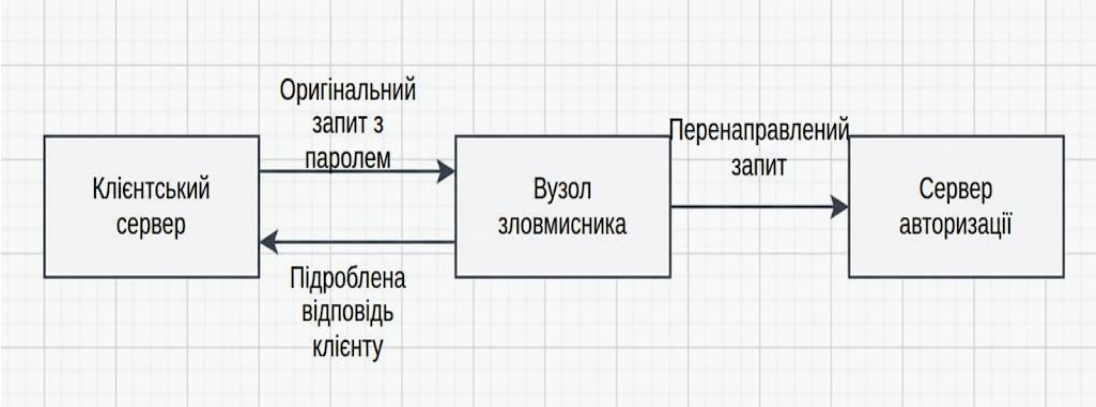

Рисунок 1.1 – Схема перехоплення трафіку.

Поряд із суто технічними дефектами, суттєве джерело загроз становлять чинники організаційного характеру. Їх виникнення безпосередньо пов'язане з ігноруванням з боку керівництва базових принципів і правил політики управління інформаційною безпекою на підприємстві, коли немає чітких політик, не проводиться незалежний аудит інформаційних систем, а оновлення для серверів розгортаються із запізненням на місяці [13]. Величезна кількість масштабних витоків даних відбувається виключно через таке недбальство та надмірні права доступу у рядових працівників. Проте найслабшою ланкою будь-

якої інфраструктури стабільно залишається людина. Користувачі масово створюють примітивні паролі, записують їх на папірцях або пересилають один одному у відкритих чатах. До того ж надзвичайну ефективність демонструє соціальна інженерія. Жертви фішингу часто самі віддають ключі доступу зловмисникам, абсолютно не розуміючи, що відбувається [7]. Іноді зливи стаються з вини інсайдерів. Це буває як навмисний продаж бази, так і випадкова помилка втомленого співробітника. Розвиток хмарних сервісів та поява тисяч розумних пристроїв лише ускладнює ситуацію. З'являється безліч нових точок входу, які хакери постійно сканують на наявність вразливостей. Структурований перелік цих загроз відображено в таблиці 1.1.

Таблиця 1.1 – Структурований перелік загроз

Категорія загрози	**Опис джерела або вразливості**
Технічні вразливості системи	Недоліки в архітектурі коду, відсутність надійного шифрування баз даних, наявність можливостей для SQL-ін'єкцій.
Мережеві перехоплення	Атаки типу Man-in-the-Middle під час передачі даних через застарілі або незахищені мережеві протоколи.
Організаційні недоліки	Ігнорування політик управління інформаційною безпекою, відсутність регулярного незалежного аудиту та затримки встановлення оновлень.
Людський фактор	Масове створення примітивних паролів, їх повторне використання на різних ресурсах, пересилання ключів доступу у відкритих чатах.
Соціальна інженерія	Проведення фішингових кампаній, внаслідок яких неуважні користувачі добровільно віддають свої ідентифікатори стороннім особам.

Таблиця демонструє багатовимірний характер загроз інформаційній безпеці, які охоплюють як технічні, так і організаційні та соціальні аспекти. Кожна категорія має власні джерела ризику, що потребують різних підходів до захисту. Загрози інформаційній безпеці формуються на перетині **технологій, організації та поведінки людей.** Технічні недоліки можна усунути через якісну розробку та аудит, мережеві ризики – через сучасні протоколи, організаційні – через політики та контроль, а людський фактор і соціальну інженерію – через навчання та підвищення обізнаності.

1.3 Джерела витоку облікових даних

Облікові дані можуть опинитися у відкритому доступі найрізноманітнішими шляхами. Іноді це результат ретельно спланованих, масштабних кібератак. А іноді – наслідок абсолютно банальної недбалості системних адміністраторів на місцях. Щоб побудувати дійсно ефективну систему моніторингу, треба спочатку зрозуміти реальні масштаби проблеми та розібрати всі можливі вектори витоків. Найпопулярнішим джерелом на сьогодні є зламані бази даних інформаційних систем. Це беззаперечний лідер [8]. Коли хакери успішно атакують сервери великих корпорацій або популярних інтернет-сервісів, вони миттєво зливають гігантські масиви інформації. До їхніх рук потрапляють мільйони записів звичайних користувачів. І це далеко не лише логіни та паролі. Разом із ними витікають точні адреси електронної пошти, номери мобільних телефонів та купа іншої персональної інформації, яка дозволяє легко ідентифікувати людину [10]. Що відбувається далі? Успішно вкрадені бази зазвичай публікуються у повністю відкритому доступі. Зловмисники роблять це або для прямого підриву репутації конкретної компанії, або для подальшої монетизації через спеціалізовані ресурси.

Специфічну роль у цьому ланцюжку відіграють тіньові інтернет-форуми. Це закриті тематичні майданчики, де щоденно відбувається активний обмін викраденою інформацією. Там хакери продають ексклюзивний доступ до найсвіжіших баз, жваво обговорюють нові технічні способи злому та просто публікують величезні архіви для всіх охочих [6]. Потрапити на такі форуми часто можна виключно за спеціальною рекомендацією.

Проте деякі з них залишаються частково відкритими навіть для звичайних пошукових систем. Ще один серйозний вектор розповсюдження компрометації – це публічні файлові сховища. Звичайні хмарні сервіси обміну файлами стали справжнім раєм для кіберзлочинців. Вони можуть роками непомітно зберігати там зашифровані архіви або звичайні текстові документи з чужими паролями.

Доступ до таких директорій видається за прямими посиланнями. Це неймовірно прискорює та спрощує масове поширення файлів серед інших зацікавлених осіб.

- Окремо варто виділити абсолютно відкриті джерела інформації, з якими працюють фахівці в рамках OSINT (розвідки на основі відкритих даних) [7]. Детальний аналіз таких платформ дозволяє виявити неочевидні ризики розповсюдження інформації [4]. Найчастіше витоки тут стаються з таких причин:
- Недосвідчені користувачі добровільно залишають в мережі інформацію, якої достатньо для часткового або повного відновлення їхніх акаунтів.
- Кваліфіковані програмісти випадково публікують власні ключі доступу у відкритих репозиторіях коду під час розробки.
- Адміністратори залишають тестові сервери без жодної автентифікації або з доступом "за замовчуванням".

Обов'язково слід враховувати ті інциденти, що відбуваються виключно через помилки в конфігураціях. Наприклад, неправильне налаштування маршрутизації або випадково залишений відкритий доступ до бази даних. У таких випадках інформація стає доступною без необхідності проведення будь-яких складних хакерських атак. У сучасних умовах значна частина подібних витоків інформації прямо пов'язана з масовим переходом бізнесу на хмарні технології. Неправильне налаштування прав доступу в хмарних середовищах миттєво призводить до публікації конфіденційної інформації. Помітно, що джерела витоків постійно еволюціонують і стають дедалі різноманітнішими. Це створює величезні перешкоди для ручного відстеження загроз і прямо обумовлює гостру необхідність розробки комплексних автоматизованих систем моніторингу.

1.4 Аналіз існуючих систем моніторингу витоків облікових даних

Ринок кібербезпеки сьогодні просто переповнений інструментами для виявлення злитих паролів. Більшість із них працюють за дуже схожим і передбачуваним принципом. Вони створюють гігантські централізовані бази даних, куди стікається інформація про всі відомі інциденти [13]. Наповнення

таких баз даних відбувається завдяки безперервній роботі спеціалізованих програм, які систематично аналізують публічні витоки, тіньові форуми, файлові сховища та інші відкриті ресурси мережі. Цей процес має характер автоматизованого моніторингу, що перетворює розрізнені джерела на структуровані масиви інформації, доступні для подальшого використання у злочинних цілях [8]. Типова схема функціонування подібних сервісів полягає в наступному: користувач переходить на вебресурс, вводить електронну пошту чи логін, після чого сервер здійснює пошук відповідних збігів у власному масиві скомпрометованих облікових записів. Такий механізм забезпечує швидку перевірку наявності даних у базах витоків та водночас демонструє масштаб проблеми, пов’язаної з централізованим накопиченням конфіденційної інформації [7]. Якщо збіг є – видається попередження. Звісно, розробники постійно намагаються прикрутити до своїх платформ нові функції. Вони додають періодичні сповіщення, вираховують рівень загального ризику для профілю, зберігають історію попередніх перевірок та видають базові поради щодо цифрової гігієни. Але суть від цього не змінюється, бо це все одно залишається класичною звіркою рядків.

За способом технічної реалізації всі ці системи можна чітко розділити на три основні категорії:

- Масові онлайн-сервіси. Це найпопулярніший варіант, оскільки він працює прямо в браузері і доступний абсолютно всім користувачам.
- Закриті корпоративні рішення. Вони глибоко інтегруються у внутрішню IT-інфраструктуру конкретних компаній і дозволяють централізовано моніторити безпеку відразу сотень співробітників.
- Спеціалізовані локальні програмні засоби.

Попри широке застосування, подібні системи мають низку концептуальних обмежень. Найбільш очевидним недоліком є неповнота інформаційних масивів: далеко не кожен витік даних стає публічним, а процес інтеграції нових баз до агрегаторів супроводжується значними часовими затримками. У результаті

користувачі отримують лише часткову картину реальної ситуації, що знижує ефективність таких інструментів у практичному застосуванні.

Іншою, ще більш серйознішою проблемою є питання безпеки та конфіденційності [1]. Щоб перевірити свій акаунт, людина змушена добровільно передати справжній ідентифікатор якомусь сторонньому невідомому серверу. А якщо цей сервіс сам стане жертвою хакерів або не має належного захисту? Тоді він перетвориться на ще один вектор атаки на користувача [9].

Окрім того, абсолютна більшість доступних платформ працює суто в ручному режимі. Поки ти сам не зробиш запит – ніхто про загрозу не попередить, що абсолютно не дозволяє оперативно реагувати на раптові витоки. Усе це переконливо доводить, що нам гостро необхідні інноваційні автоматизовані інструменти. Вони мають забезпечувати безперервний моніторинг без ризику додаткової компрометації самих даних.

1.5 Аналіз загроз та ризиків, пов'язаних із витоком облікових даних

Витік автентифікаційних даних не можна розглядати як ізольовану подію. Він стає відправною точкою для цілого комплексу взаємопов’язаних проблем, що впливають як на життєдіяльність окремої особи, так і на функціонування великих корпоративних структур. Наслідки подібної компрометації мають багатовимірний характер і проявляються у трьох ключових площинах: технічній, економічній та репутаційній. Детальну класифікацію цих загроз та їхній вплив на інфраструктуру відображено у табл. 1.2.

Таблиця 1.2 – Класифікація загроз та їхній вплив на інфраструктуру

Тип наслідків	**Характер впливу на інфраструктуру**
Технічні наслідки	Прямий несанкціонований доступ до внутрішніх систем, повне захоплення управління профілем, використання акаунта як плацдарму для автоматизованих атак.
Економічні наслідки	Несанкціоновані грошові транзакції, пряма крадіжка коштів з рахунків, колосальні фінансові збитки від корпоративного шпигунства.

Юридичні наслідки	Необхідність виплати штрафів за порушення законодавства у сфері захисту персональних даних, проведення тривалих судових розглядів.
Репутаційні наслідки	Публічне розголошення факту компрометації, стрімке зниження рівня довіри користувачів, масовий відтік клієнтів до конкурентів.

Однією з найбільш критичних технічних проблем є прямий несанкціонований доступ до інформаційних систем. Зазвичай це супроводжується повним і безповоротним захопленням управління профілем справжнього власника. Ситуація, коли кіберзлочинець отримує чинні автентифікаційні дані – коректний логін та пароль, що відкриває йому можливість діяти від імені легітимного користувача. У такому випадку стандартні периметри мережевого захисту підприємства фактично втрачають свою ефективність, адже доступ здійснюється у повністю легальний спосіб з точки зору системи. Це створює передумови для неконтрольованого проникнення у внутрішні ресурси та подальшої ескалації атак. [9].

Після компрометації облікового запису зловмисник отримує можливість виконувати будь-які деструктивні дії від імені легітимного користувача. Це створює прямий канал для прихованого викрадення конфіденційної персональної інформації або несанкціонованої модифікації критично важливих системних параметрів. У такому випадку захоплений профіль перетворюється на стійкий плацдарм для подальших цілеспрямованих атак на інших учасників локальної мережі.

Особливу небезпеку становить тенденція до масового використання сучасними хакерами спеціалізованого програмного забезпечення, яке автоматизує процеси пошуку вразливостей та здійснення атак. Це значно підвищує ефективність злочинних дій і ускладнює їх виявлення традиційними методами захисту. Алгоритм роботи такого шкідливого програмного забезпечення проілюстровано в Додатку А.

Воно необхідне для автоматизованої перевірки вкрадених ідентифікаторів на сотнях інших популярних ресурсів. У комп'ютерній інженерії це відомо як

атаки повторного використання паролів. Оскільки люди мають дуже шкідливу звичку використовувати однакові комбінації символів абсолютно всюди, один непомітний витік з якогось маловідомого ігрового форуму має катастрофічні наслідки. Він миттєво призводить до невідворотної втрати доступу до корпоративної робочої пошти, державних електронних порталів чи банківських додатків [8].

Якщо відкинути суто технічні аспекти, компрометація ідентифікаційних даних тягне за собою надзвичайно болючі економічні та юридичні наслідки для всіх сторін інциденту. Фінансові ризики зазвичай мають прямий зв'язок з несанкціонованими грошовими транзакціями або крадіжкою коштів з рахунків. У випадку корпоративного сектору додаються ще й колосальні збитки від промислового шпигунства. Графік динаміки зростання фінансових втрат від подібних інцидентів за останні роки наведено на рисунку 1.2.

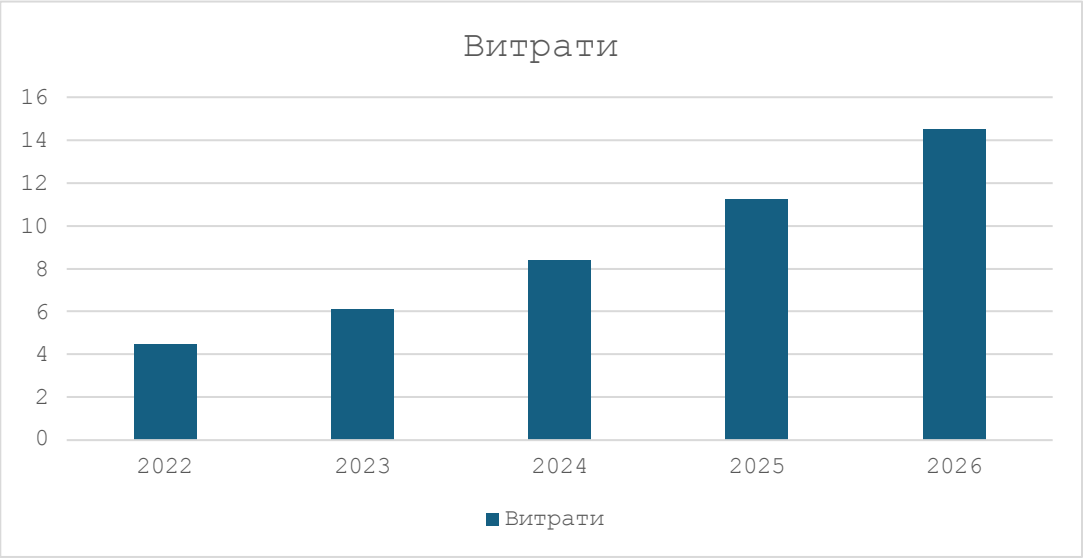

Рисунок 1.2 – Динаміка зростання фінансових втрат від компрометації облікових даних

Юридичні наслідки для організацій, які через власну недбалість допустили витік інформації своїх клієнтів, просто розгромні. Це і необхідність виплати величезних штрафів за порушення суворого законодавства у сфері

захисту персональних даних [10], і проведення тривалих, шалено дорогих судових розглядів [11].

Не менш руйнівними є й репутаційні ризики. Публічне розголошення того факту, що компанія виявилася нездатною захистити інформацію власних користувачів, гарантовано призводить до стрімкого зниження рівня довіри. Клієнти масово йдуть до конкурентів. Компанія втрачає свої позиції на ринку. Ступінь тяжкості всіх цих негативних наслідків не є статичним. Він прямо пропорційно залежить від одного єдиного фактора: наскільки швидко користувач або профільна служба безпеки дізнається про факт витоку. Якщо реакція швидка, можна встигнути вжити контрзаходи – терміново змінити паролі або повністю заблокувати скомпрометовані рахунки.

Саме цей критичний фактор втраченого часу є беззаперечним аргументом. Він підтверджує абсолютну та гостру необхідність термінового впровадження систем безперервного проактивного моніторингу. Нам потрібні інструменти, які здатні автоматично виявляти приховані загрози на ранніх етапах, тим самим зводячи потенційні збитки до абсолютного мінімуму [4].

1.6 Формування вимог до системи моніторингу витоку облікових даних

На основі проведеного глибокого аналізу глобальної проблеми витоку облікових даних, джерел їх постійного поширення, а також критичного огляду існуючих систем моніторингу виникає цілком об'єктивна інженерна необхідність формування чіткого переліку вимог до проєктуємого програмного засобу. Структурований перелік цих вимог наведено в таблиці 1.3. Усі ці сформовані вимоги традиційно поділяються на три великі взаємопов'язані категорії. Це функціональні, нефункціональні та суворі вимоги до інформаційної безпеки. Функціональна частина безпосередньо визначає базову поведінку майбутньої системи та ті конкретні задачі, які вона повинна успішно виконувати під час роботи.

Таблиця 1.3 – Структурований перелік вимог

Тип вимоги	**Опис вимоги**
Функціональні вимоги	Автоматизований прийом запитів через месенджер, сувора валідація інформації, взаємодія з зовнішніми джерелами та формування зрозумілого звіту з превентивними рекомендаціями.
Нефункціональні вимоги	Висока швидкодія при обробці паралельних запитів, відмовостійкість під час пікових навантажень, модульність архітектури та максимальна зручність для користувача.
Вимоги до безпеки	Сувора заборона на збереження діючих паролів, використання захищених криптографічних каналів та повна відсутність прихованих логів із персональними запитами.

Розроблювана система має обов'язково забезпечувати повністю автоматизований прийом вхідних запитів від кінцевого користувача. Це має відбуватися виключно через зручний діалоговий інтерфейс месенджера. Наступним критичним кроком є автоматична синтаксична обробка та сувора валідація отриманої інформації для негайного відсіювання некоректних або відверто зловмисних запитів. Після успішної валідації програмний засіб повинен самостійно ініціювати пошук інформації про можливі витоки. Робиться це шляхом складної асинхронної взаємодії із зовнішніми довіреними джерелами або спеціалізованими агрегаторами кіберінцидентів через прикладні програмні інтерфейси. Також вкрай необхідною функцією є глибокий аналіз отриманих результатів для адекватної оцінки поточного рівня ризику та подальше інформування користувача у максимально зрозумілому вигляді з наданням чітких превентивних рекомендацій щодо подальших дій.

Нефункціональні вимоги, у свою чергу, докладно описують загальні атрибути якості системи та ті жорсткі обмеження, за яких вона повинна стабільно функціонувати в реальних умовах експлуатації. До таких вимог насамперед належить висока швидкодія [3]. Вона передбачає здатність серверної частини системи одночасно обробляти велику кількість паралельних запитів від абсолютно різних користувачів без виникнення значних затримок у мережі. Надійність та відмовостійкість системи вимагають забезпечення стабільної роботи навіть під час екстремальних пікових навантажень. Також сюди входить

вимога щодо коректної обробки будь-яких програмних помилок, що можуть раптово виникати при тимчасовій недоступності зовнішніх баз даних.

Надзвичайно важливою вимогою є модульність та масштабованість архітектури системи. Структурну діаграму взаємодії цих модулів можна переглянути на рис. 1.3.

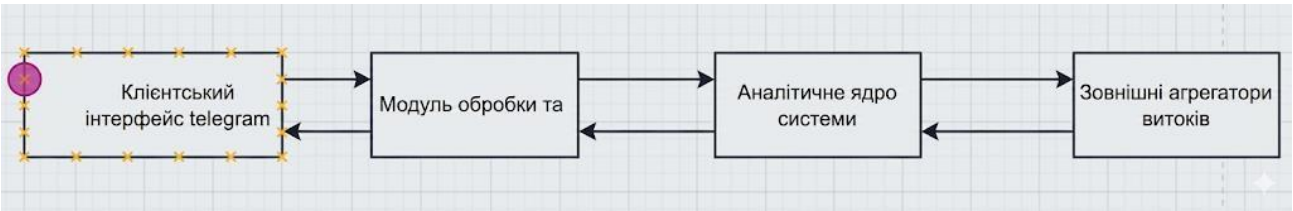

Рисунок 1.3 – Структурна діаграма взаємодії модулів системи.

Саме модульний підхід дозволяє легко оновлювати окремі компоненти програми, наприклад модуль інтеграції або лінгвістичний аналізатор, без необхідності повної зупинки всього сервісу. Окрім цього, система повинна відрізнятися максимальною зручністю для кінцевого користувача, що досягається саме за рахунок повної відмови від складних, перевантажених веб-інтерфейсів на користь звичного та інтуїтивно зрозумілого середовища популярного месенджера.

Вимоги щодо інформаційної безпеки є абсолютно критичними для проєктів подібного класу, оскільки система працює з потенційно чутливою персональною інформацією, що може бути легко перехоплена зловмисниками. Головним і непорушним правилом проєктування в даному випадку є сувора заборона на збереження будь-яких діючих паролів або відкритих ідентифікаторів користувачів у внутрішніх базах даних системи [1]. Програмний засіб суворо зобов'язаний використовувати виключно захищені криптографічні канали передачі інформації при будь-якій взаємодії із зовнішніми сервісами та гарантувати повний захист від несанкціонованого доступу до серверного обладнання. Крім того, архітектура повинна бути побудована таким чином, щоб повністю унеможливити ведення будь-яких прихованих логів із персональними

запитами користувачів. Тільки це гарантуватиме їхню абсолютну конфіденційність та повну відповідність сучасним міжнародним стандартам обробки цифрових даних [10]. Тільки беззаперечне виконання всього цього комплексу сформованих вимог дозволить нам створити дійсно надійний та високоефективний інструмент проактивного захисту.

Підсумовуючи, варто зазначити, що у першому розділі кваліфікаційної роботи було максимально детально розглянуто теоретичні основи проблеми витоку облікових даних та проведено комплексний системний аналіз факторів, що безпосередньо впливають на їх компрометацію. Було встановлено абсолютно критичне значення облікових даних для стабільного функціонування сучасних цифрових сервісів та глибоко досліджено технічні, організаційні і складні поведінкові причини виникнення масштабних витоків. Нам вдалося визначити основні мережеві джерела поширення скомпрометованої інформації, серед яких найбільшу загрозу сьогодні становлять зламані корпоративні бази, закриті тіньові хакерські форуми та неправильно налаштовані хмарні сховища [8].

Також було проаналізовано поточний стан існуючих на ринку систем моніторингу, що дозволило чітко виявити їхні серйозні концептуальні обмеження, зокрема неповноту агрегованих даних та наявність суттєвих ризиків для конфіденційності кінцевих користувачів [7]. На основі цієї всебічної оцінки загроз і ризиків було сформовано деталізовані функціональні, нефункціональні та безпекові вимоги до проєктуємої системи моніторингу витоку облікових даних. Ці сформовані вимоги створюють потужну та науково обґрунтовану теоретичну інженерну базу для переходу до наступного етапу роботи, а саме до безпосереднього проєктування модульної архітектури та розробки конкретних алгоритмів функціонування майбутнього програмного засобу.

РОЗДІЛ 2 СИСТЕМА МОНІТОРИНГУ ВИТОКУ ОБЛІКОВИХ ДАНИХ

Цей розділ повністю присвячений вирішенню комплексних інженерних питань. Мова йде про безпосереднє проєктування внутрішньої структури, модульної бази та загальної архітектури нашої автоматизованої системи, яка шукатиме злиті облікові дані по всіх відкритих мережевих джерелах. Ми спираємося на вже сформовані функціональні, нефункціональні та суворі безпекові вимоги. Саме вони диктують загальну концепцію. Програмний засіб буде побудовано із жорстким застосуванням виключно сучасних підходів комп'ютерної інженерії. Тут формується максимально розгорнутий опис усіх внутрішніх логічних модулів серверної частини схематичне відображення цих компонентів та їхніх зв'язків наведено в рис. 2.1.

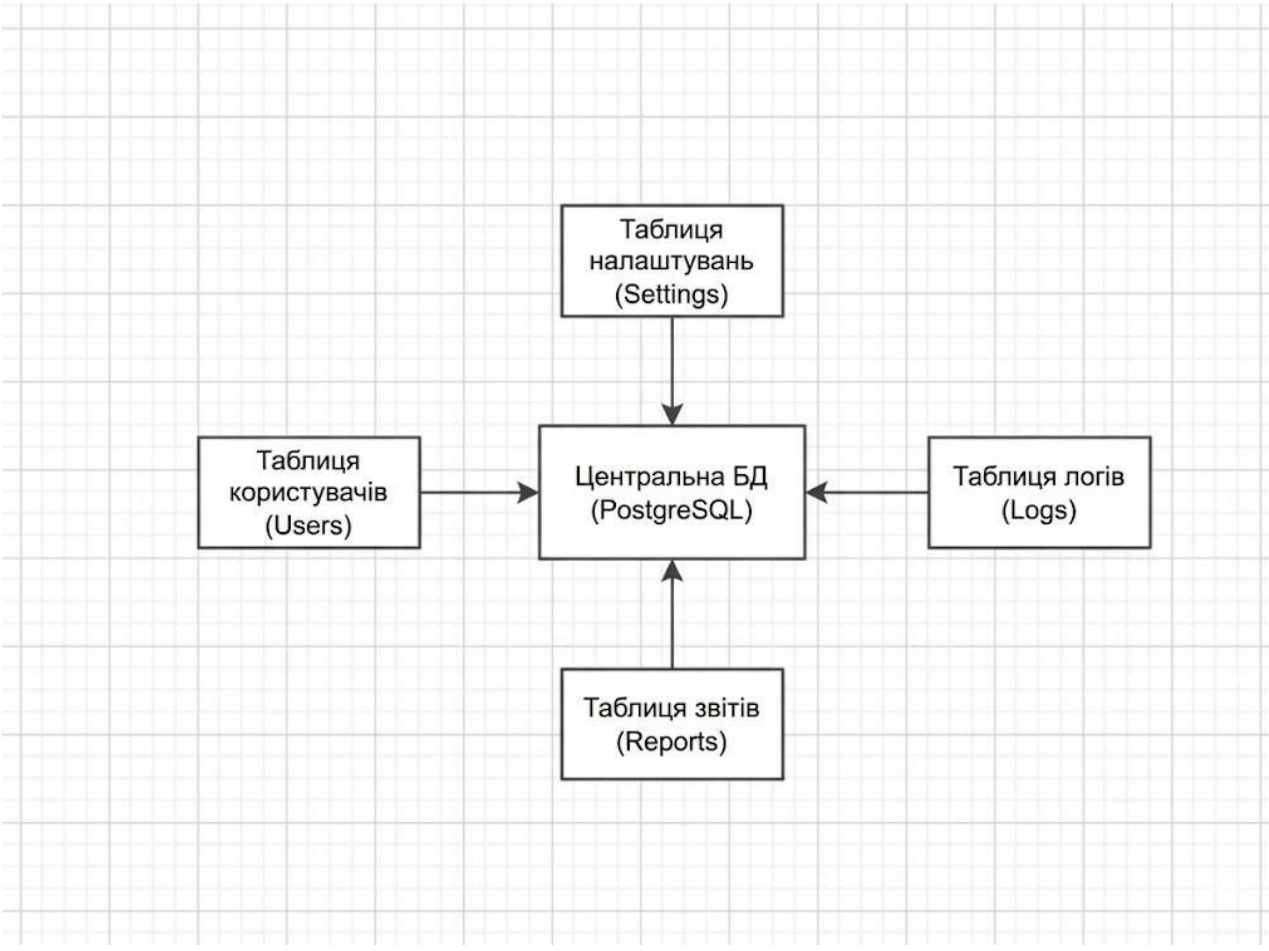

Рисунок 2.1 – Архітектура бази даних системи.

Також запропоновано чіткий покроковий алгоритм. Він описує безперервне функціонування нашої розробки в режимі реального часу. Окремо і

дуже детально обґрунтовано концептуальні підходи щодо безпечної інтеграції системи із зовнішніми віддаленими API для отримання даних [12].

Величезну увагу приділено науковому обґрунтуванню клієнтського середовища. Чому саме архітектура діалогового інтерфейсу в месенджері є найбільш оптимальною та безпечною? Відповідь досить проста [13]. Цей підхід гарантує миттєву взаємодію звичайного користувача з обчислювальним ядром. Немає жодної потреби розробляти чи роками підтримувати класичні громіздкі веб-сайти. Десктопні програми теж відпадають. Таке рішення ідеально лягає в русло сучасних світових тенденцій створення легковагових клієнт-серверних застосунків у сфері інформаційної безпеки [5].

2.1 Загальна концепція системи

Розроблювана система концептуально проєктується як сучасний автоматизований інструмент інформаційної безпеки. Її головне призначення полягає в тому, щоб максимально оперативно виявляти достовірні факти компрометації ідентифікаторів у різноманітних відкритих мережевих джерелах. Традиційні інженерні підходи до розв'язання цієї проблеми зазвичай зводяться до банального створення ізольованих класичних веб-сайтів або дуже важких десктопних додатків [5]. Однак такі архітектурні рішення суттєво знижують загальну оперативність взаємодії. Вони створюють абсолютно непотрібні бар'єри для кінцевого користувача. Щоб здійснити базову перевірку, людина змушена переходити на якісь сторонні ресурси, проходити нудні процедури реєстрації або встановлювати спеціалізоване програмне забезпечення. В умовах постійного зростання кіберзагроз це викликає цілком обґрунтовану підозру та недовіру з точки зору збереження персональної конфіденційності [1].

На противагу цим застарілим методам, ми поклали в саму основу архітектури інноваційну концепцію діалогового сервісного агента (загальну структурну схему цієї взаємодії дивіться в рис. 2.2.

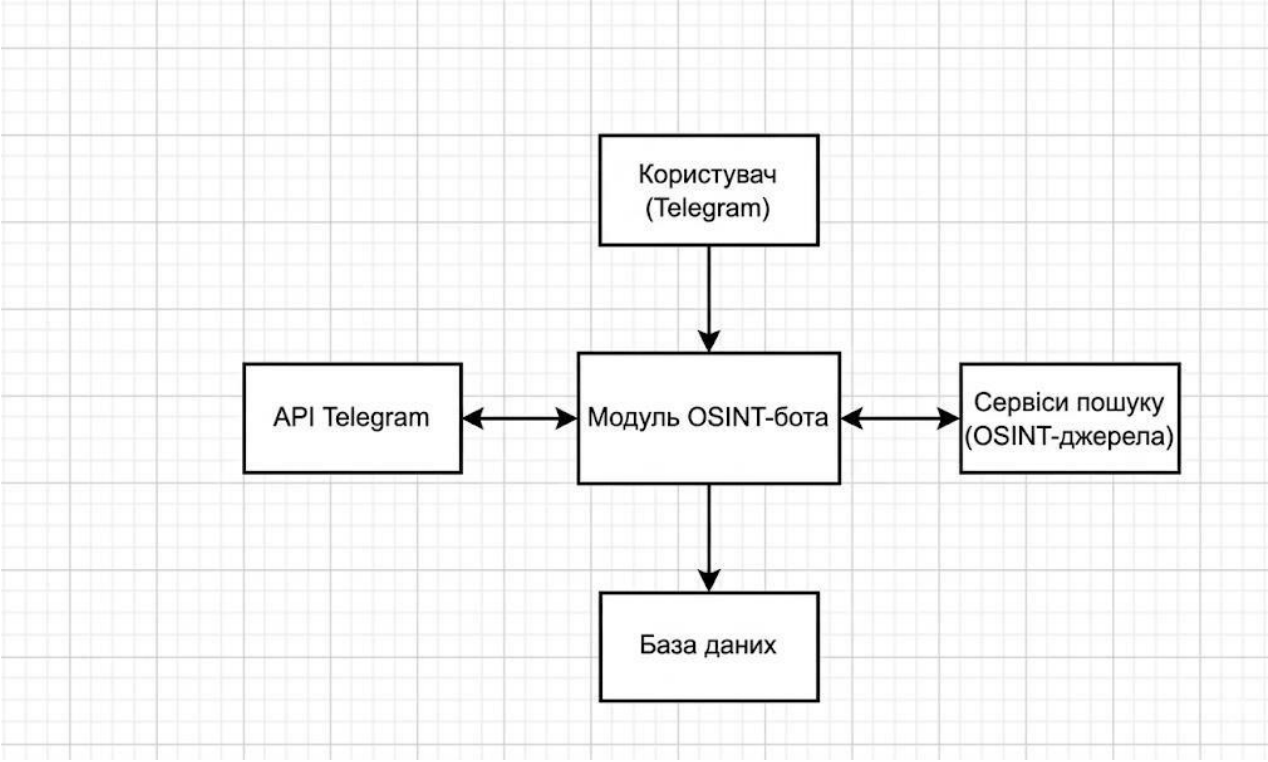

Рисунок 2.2 – Схема інформаційної взаємодії компонентів системи.

Це асинхронний інформаційний бот [13]. Він постійно і безперебійно функціонує безпосередньо всередині популярного середовища обміну повідомленнями.

Використання сучасного месенджера виконує тут роль стандартизованого, криптографічно захищеного та абсолютно кросплатформеного клієнтського середовища. Такий продуманий інженерний підхід має колосальну перевагу, оскільки він дозволяє радикально мінімізувати будь-які вимоги до апаратних ресурсів кінцевого пристрою. Вся важка обчислювальна логіка повністю виноситься на ізольований серверний рівень [11]. Окрім цього, гарантується миттєва доставка критичних сповіщень про стан безпеки прямо на мобільний телефон або комп'ютер у режимі реального часу. Процес перевірки стає максимально природним та непомітним. З інженерної точки зору спроєктована система працює як гнучкий та швидкий інтелектуальний шлюз. Вона виступає посередником між звичним графічним інтерфейсом чату та складними глобальними реєстрами компрометацій. Серверна частина приймає текстовий запит і здійснює його первинну сувору фільтрацію. Це робиться для миттєвого виявлення шкідливих ін'єкцій або звичайних синтаксичних помилок. Потім

система самостійно транслює цей запит до зовнішніх розподілених баз даних через захищені мережеві канали зв'язку [12]. Після отримання відповіді від глобальних спеціалізованих сервісів агрегації виконується глибокий аналіз отриманих сирих пакетів даних. Фінальний етап – це швидка зворотна доставка вже повністю структурованого аналітичного звіту приклад такого звіту можна побачити в таблиці 2.1.

Параметр звіту	**Значення та опис**
Ідентифікатор	Електронна пошта або логін, що перевіряється системою на факт компрометації.
Статус інциденту	Підтвердження знайденого витоку або повідомлення про відсутність збігів у базах.
Джерело компрометації	Конкретні назви зламаних онлайн-платформ та точні дати зафіксованих інцидентів безпеки.
Скомпрометовані дані	Перелік конкретних викрадених полів, наприклад паролі, хеші чи просто нікнейми.
Превентивні інструкції	Індивідуальні рекомендації щодо зміни паролів та активації засобів багатофакторного захисту.

Таблиця 2.1 – Приклад структурованого аналітичного звіту

Цей звіт обов'язково супроводжується комплексом індивідуальних рекомендацій щодо подальшої мінімізації виявлених кіберризиків.

2.2 Методи та підходи до боротьби з витоками даних

У сучасній практиці комп'ютерної інженерії для забезпечення надійного захисту мало просто констатувати факт інциденту. Потрібен цілий комплекс спеціалізованих методів обробки даних, щоб оперативно виявляти скомпрометовану інформацію та мінімізувати її руйнівні наслідки. В основу нашої розробки ми поклали метод превентивної сигнатурної перевірки. Це означає глибокий синтаксичний аналіз. Він забезпечує сувору первинну

фільтрацію всього вхідного потоку ще до того, як запит піде у зовнішню мережу. На практиці це виглядає як автоматична звірка введених користувачем даних із жорстко заданими регулярними виразами для поштових скриньок чи логінів. Такий підхід надійно відрізає спроби експлуатації вразливостей (наприклад, через ін'єкції специфічних символів) і відчутно знижує непотрібне навантаження на мережеві шлюзи [9].

Наступний ключовий підхід – це метод безпечної інтеграції з розподіленими реєстрами. Ми реалізуємо його через стандартизовані REST API виклики за допомогою асинхронних бібліотек [4].

Його головна архітектурна перевага надзвичайно проста, але ефективна. Система здійснює абсолютно ізольований транзитний запит до зовнішніх агрегаторів на кшталт Have I Been Pwned [7]. При цьому немає жодної необхідності передавати кудись або зберігати у власних проміжних базах дійсні паролі. Це стовідсотково гарантує одну дуже важливу річ. Навіть якщо весь мережевий трафік буде теоретично перехоплено, сам процес нашого моніторингу ніколи не перетвориться на додаткове джерело витоку чутливих ідентифікаторів.

Що відбувається після успішного отримання відповіді від зовнішніх джерел? Система відразу застосовує метод комплексної оцінки критичності виявленого інциденту. Цей аналітичний підхід полягає в детальному парсингу структури знайденого витоку (зазвичай це розбір сирої JSON-відповіді від сервера). Далі йде автоматична класифікація рівня ризику. Тут усе залежить від того, які саме типи даних постраждали. Алгоритмічно система чітко розмежовує ситуації. Одне діло, коли у відкритий доступ потрапив лише публічний логін. І зовсім інше – дійсно критичні зливи, де відбулася повна компрометація пошти разом із текстовим паролем або MD5-хешами. Логічним завершенням усього процесу обробки є метод превентивного формування рекомендацій. На основі щойно отриманого профілю ризику скрипт генерує для кінцевого користувача максимально зрозумілий алгоритм дій і відправляє його через інтерфейс месенджера [13]. Цей текст зазвичай містить суворі інструкції щодо негайної зміни автентифікаційних маркерів на всіх пов'язаних платформах. Також туди

входять рекомендації примусово завершити всі активні поточні сесії та обов'язково активувати засоби багатофакторного захисту. Саме завдяки такому вдалому поєднанню глибоких аналітичних та превентивних методів, розроблювана система перетворюється з простого інструменту пасивного спостереження на повноцінний автоматизований засіб активної боротьби з наслідками мережевих витоків.

2.3 Розробка архітектури системи

Архітектура нашої розроблюваної системи чітко визначає її фундаментальну інженерну структуру. Вона описує характер взаємодії між ключовими компонентами та загальні принципи обробки інформаційних потоків. Програмний засіб проєктується виключно за класичним багаторівневим клієнт-серверним принципом. Тут усі функціональні обов'язки дуже жорстко розподілені між трьома основними логічними рівнями. Навіщо це потрібно? Це робиться для забезпечення максимальної модульності, стійкості до високих мережевих навантажень та дотримання базових стандартів інформаційної безпеки. Якщо розглядати клієнтський рівень системи, то він повністю реалізується на базі існуючого графічного діалогового інтерфейсу месенджера [13]. Цей рівень відповідає виключно за безпосередню взаємодію з кінцевим користувачем. Він збирає текстове введення у вигляді керівних команд або конкретних перевірочних ідентифікаторів. Після цього клієнтська частина просто відображає фінальні результати пошуку у зручному, інтуїтивно зрозумілому візуальному форматі. Головна інженерна перевага такого підходу цілком очевидна. Клієнтський рівень взагалі не виконує жодних складних аналітичних обчислень і принципово не зберігає локальних даних перевірки. Це робить його абсолютно захищеним від можливих компрометацій безпосередньо на боці мобільного пристрою чи персонального комп'ютера користувача.

Перейдемо до серверного рівня. Він являє собою головне обчислювальне логічне ядро всієї нашої OSINT-системи. Цей модуль розгортається на спеціалізованій ізольованій апаратній або ж хмарній платформі. Цей

центральний вузол виконує надзвичайно важливу роботу. Він забезпечує безперервний асинхронний прийом усіх вхідних мережевих запитів від клієнтського інтерфейсу [5]. Отримавши дані, ядро відразу здійснює їх глибоку синтаксичну обробку та обов'язкову безпекову валідацію. Крім того, саме серверний рівень повністю несе відповідальність за формування та правильну маршрутизацію вихідних запитів до зовнішніх інформаційних ресурсів. Сервер парсить отриману сиру інформацію та алгоритмічно генерує кінцеву структуровану відповідь для користувача.

Третім критично важливим компонентом загальної архітектури є зовнішній рівень. Концептуально він складається з абсолютно незалежних глобальних сервісів агрегації кіберінцидентів та відкритих баз даних [8]. Взаємодія між розроблюваним серверним ядром та цим зовнішнім рівнем здійснюється суворо через захищені протоколи передачі даних. Для цього використовуються стандартизовані прикладні програмні інтерфейси [7]. Що це дає на практиці? Це забезпечує дуже надійну ізоляцію внутрішньої бізнес-логіки системи від будь-яких мережевих загроз із боку сторонніх ресурсів. Такий чіткий і продуманий архітектурний розподіл на окремі незалежні рівні дозволяє максимально ефективно модернізувати та масштабувати окремі вузли програмного засобу в майбутньому. І робиться це без жодного ризику порушення його загальної стабільності роботи.

2.4 Структура системи та характеристика основних модулів

Внутрішня структура серверної частини нашого розробленого Defensive OSINT бота побудована за суворим блочно-модульним принципом. Це абсолютно класичний підхід у сучасній комп'ютерній інженерії для забезпечення високої надійності, стійкості до відмов та можливості подальшого масштабування. Кожен компонент цієї структури виконує свою суворо визначену, ізольовану функцію. Обмін інформацією між ними відбувається виключно через регламентовані внутрішні інтерфейси. Першим і найголовнішим вузлом на шляху вхідних даних виступає модуль взаємодії та диспетчеризації команд. Якщо

говорити практично, його основне інженерне завдання полягає у встановленні та підтримці сесій з клієнтським інтерфейсом Telegram через розпізнавання унікальних chat_id кожного користувача [13]. Цей модуль безперервно слухає вхідні текстові повідомлення. Він здійснює їх первинний логічний розподіл залежно від типу отриманої інформації. Наприклад, він чітко відокремлює системні команди ініціалізації (типу /start або /help) від безпосередніх користувацьких ідентифікаторів, які вже підлягають перевірці на витік. Після етапу диспетчеризації в роботу автоматично вмикається модуль сигнатурної валідації та безпеки. Він фактично відіграє роль внутрішнього брандмауера та ключового фільтра всієї нашої системи. Модуль виконує жорстку нормалізацію отриманих рядків. Він перевіряє їх на повну відповідність встановленим синтаксичним шаблонам, використовуючи регулярні вирази для поштових скриньок чи допустимих форматів логінів. Якщо цей скрипт виявляє порушення структури або бачить потенційні ознаки спроби ін'єкції шкідливого коду, він миттєво перериває подальший рух запиту. Системі повертається сигнал про помилку введення. Це надійно захищає наступні логічні рівні від виконання некоректних операцій та падіння самого процесу.

У випадку успішного проходження валідації очищені дані передаються далі. Вони потрапляють до модуля інтеграції та трансляції запитів, який безпосередньо виконує критично важливі функції захищеного мережевого шлюзу.

Цей компонент відповідає за всю взаємодію внутрішньої логіки із зовнішніми віддаленими джерелами. Він інкапсулює перевірені дані у стандартизовану структуру захищеного мережевого запиту. Далі встановлюється асинхронне з'єднання з глобальними агрегаторами витоків (наприклад, через виклики з використанням бібліотеки aiohttp до REST API платформ OSINT Framework) [4, 8]. Модуль забезпечує коректну обробку специфічних мережевих проблем. Йдеться про боротьбу з лімітами запитів (rate limits) від серверів, обробку можливих мережевих затримок чи раптових обривів зв'язку. Після цього здійснюється парсинг отриманих сирих JSON-пакетів відповідей у внутрішні

змінні нашої системи. Завершальним етапом усього життєвого циклу керує модуль формування аналітичних звітів. Він бере на себе абсолютно всю інтелектуальну складову аналізу. Модуль приймає вже розшифровані масиви від інтеграційного блоку, детально інтерпретує їх та вираховує загальний індекс критичності знайденої компрометації. При цьому враховується давність конкретного інциденту, тип зламаної платформи та повний перелік викрадених полів (паролі, хеші чи просто нікнейми). На основі проведених аналітичних обчислень генерується фінальний структурований текстовий звіт. Цей звіт доповнюється необхідними превентивними інструкціями з кібербезпеки та передається назад до модуля диспетчеризації для миттєвого відправлення в діалоговий чат користувача. Такий глибокий розподіл на незалежні модулі гарантує безперебійну роботу бота та дозволяє легко оновлювати його код без зупинки моніторингу [11].

2.5 Алгоритм функціонування системи

Життєвий цикл обробки одного інформаційного запиту в розроблюваній системі описується дуже чіткою та послідовною зміною логічних етапів. Усе починається з того моменту, коли кінцевий користувач ініціює нову сесію в діалоговому інтерфейсі. Він відправляє спеціальну стартову команду (як правило, це стандартний /start для платформи месенджера). Після цього система автоматично ініціалізує необхідні внутрішні змінні та переходить у стан активного очікування. Бот чекає на введення цільових даних [13]. Коли користувач надсилає текстовий ідентифікатор – наприклад, адресу своєї персональної електронної пошти або корпоративний логін, – цей рядок миттєво перехоплюється диспетчером команд. Далі він безпосередньо передається до модуля валідації [3]. На цьому критичному етапі алгоритм передбачає жорстку перевірку синтаксису через регулярні вирази. Якщо введена інформація не відповідає заздалегідь визначеним шаблонам, процес перевірки негайно переривається. У такому випадку системний алгоритм генерує повідомлення про помилку формату. Він відправляє його користувачеві як коротку підказку.

Програмний засіб одразу повертається у вихідний стан очікування коректного введення. Помилковий запит повністю ігнорується і знищується з пам'яті.

Що відбувається у випадку успішного проходження валідації? Перевірені очищені дані автоматично надсилаються до модуля інтеграції. Він інкапсулює їх у безпечний асинхронний HTTP-запит до віддаленого прикладного програмного інтерфейсу глобального агрегатора кіберінцидентів [7]. Алгоритм функціонування системи на цьому транзитному етапі передбачає одну вкрай важливу річ. Це обов'язкова обробка можливих мережевих тайм-аутів або непередбачуваних помилок доступу на стороні зовнішнього сервера. З'єднання може раптово розірватися. Віддалена база даних може бути тимчасово недоступною або повернути помилку ліміту запитів. Якщо стається щось подібне, алгоритм активує сценарій безпечного переривання поточної операції [5]. У чат виводиться відповідне інформаційне повідомлення про тимчасову недоступність сервісу. Це стовідсотково гарантує відсутність критичних збоїв або повного зависання в роботі самого обчислювального ядра.

Після успішного прийому інформаційного пакету від зовнішнього ресурсу ініціюється фінальна стадія. Це етап парсингу та аналітики. Якщо зовнішня база даних повертає порожній результат (наприклад, порожній масив), алгоритм фіксує відсутність відомих компрометацій. Бот відразу формує відповідний заспокійливий звіт. Ситуація кардинально змінюється, якщо у отриманій JSON-відповіді містяться підтверджені збіги [11]. Модуль аналітики детально розбирає отриманий масив інформації. Він алгоритмічно виділяє конкретні назви зламаних онлайн-платформ і точні дати зафіксованих інцидентів. Обов'язково витягується перелік конкретно скомпрометованих полів даних (паролі, хеші чи просто нікнейми). Після цього до структурованого звіту додаються превентивні інструкції з кібербезпеки, що відповідають виявленому рівню загрози [2]. Згенерований фінальний текст автоматично надсилається через платформу месенджера прямо на екран користувачеві. На цьому життєвий цикл поточного запиту успішно завершується. Система скидає стан і повертається до штатного режиму очікування нових команд.

2.6 Інтеграція із зовнішніми джерелами інформації

Ефективність функціонування нашої системи повністю базується на продуманих методах автоматизованої взаємодії із зовнішніми реєстрами. Роль таких реєстрів виконують спеціалізовані глобальні сервіси агрегації кіберінцидентів та відкриті бази скомпрометованих даних [8]. В сучасних умовах така інтеграція реалізується виключно через стандартизовані прикладні програмні інтерфейси, побудовані на принципах REST (Representational State Transfer) архітектури. Використання цієї моделі дозволяє нам повністю автоматизувати обмін даними. Це гарантує високу швидкість обробки запитів, адже ми отримуємо "чисту" аналітику прямо від джерела, не витрачаючи ресурси на локальне зберігання гігабайтних масивів тіньової інформації на власному сервері. Для гарантованого доступу до зовнішніх API наша система використовує унікальні криптографічні маркери (API-ключі). Вони передаються в заголовках кожного вихідного запиту, щоб віддалений сервер міг підтвердити легітимність нашого шлюзу.

Окремим, надзвичайно важливим аспектом є забезпечення тотального захисту транзитних даних. Будь-яка передача інформації здійснюється суворо через захищені канали зв'язку із застосуванням сучасних протоколів наскрізного шифрування (TLS) [1]. При формуванні мережевого запиту до віддалених сервісів система штучно обмежує обсяг даних. Ми відправляємо лише мінімально необхідні ідентифікатори – це повністю виключає можливість витоку паролів або інших чутливих параметрів у процесі виконання перевірки.

Окрім того, для підтримки стабільності в умовах пікових навантажень, ми реалізували спеціальні алгоритмічні підходи. Зовнішні постачальники даних завжди встановлюють жорсткі обмеження (rate limits) на кількість запитів, щоб захиститися від DDoS-атак [12]. Щоб наш бот не потрапив під бан, було впроваджено механізми асинхронного очікування відповідей. Це досягається завдяки динамічному регулюванню частоти відправлення пакетів (так званий

"backoff algorithm"). Подібна стратегія гарантує безперебійну роботу розробленого засобу та повністю унеможливлює його автоматичне блокування з боку сторонніх серверів кібербезпеки, навіть при інтенсивному потоці користувачів.

2.7 Обґрунтування інженерних рішень та архітектурного підходу

Обґрунтування обраного архітектурного підходу є критично важливим етапом проєктування, адже саме воно підтверджує доцільність прийнятих інженерних рішень та їх відповідність вимогам технічного завдання. Наша концепція, яка гармонійно поєднує діалоговий інтерфейс месенджера з віддаленими мережевими шлюзами на базі REST архітектури, дозволяє ефективно вирішити одразу кілька складних завдань [5]. Насамперед цей підхід забезпечує повне логічне розділення зон відповідальності між різними компонентами, що в інженерній практиці відомо як принцип максимального послаблення зв'язностей. Серверна обчислювальна частина системи виявляється фізично та логічно ізольованою від інтерфейсу користувача та від масивних зовнішніх баз даних. Така сувора ізоляція кардинально спрощує процес модернізації програмного коду, проведення профілактичного обслуговування, модульного тестування та розширення функціональності окремих вузлів [11]. Ми не ризикуємо порушити стабільність всього комплексу при оновленні лише одного модуля. Окрім того, відсутність потреби підтримувати складні веб-інтерфейси суттєво економить час розробника, дозволяючи зосередитися виключно на вдосконаленні алгоритмів безпеки.

Не менш важливою перевагою прийнятого рішення є глибока оптимізація споживання апаратних ресурсів серверного обладнання. Завдяки повній відмові від локального зберігання гігабайтних масивів скомпрометованих баз даних та переходу на легковагові транзитні запити, загальні вимоги до дискової та оперативної підсистеми знижуються до абсолютного мінімуму [10]. Це дозволяє стабільно розгортати систему на базових віртуальних хмарних серверах без жодної втрати швидкодії. Але найголовнішим аргументом є забезпечення

безпрецедентного рівня інформаційної безпеки. Оскільки наш продукт принципово не зберігає дійсні паролі у внутрішні таблиці та свідомо не накопичує довгострокову історію перевірок із прив'язкою до осіб, навіть гіпотетична компрометація самого сервера не несе жодної реальної загрози для клієнтів. Такий архітектурний дизайн стовідсотково відповідає найсуворішим міжнародним стандартам безпечного проєктування [9].

Підсумовуючи другий розділ кваліфікаційної роботи, можна впевнено стверджувати, що ми детально розглянули всі ключові аспекти архітектурного проєктування нашої системи моніторингу витоків. На основі системного аналізу було сформовано та обґрунтовано концепцію, яка базується на інтеграції месенджера замість громіздких веб-рішень. Було доведено, що такий підхід забезпечує максимальну оперативність взаємодії та суттєво вищий рівень довіри з боку користувачів. Ми успішно розробили надійну багаторівневу модульну архітектуру серверної частини, яка включає ізольовані компоненти для диспетчеризації команд, синтаксичної валідації, безпечної інтеграції із зовнішніми ресурсами та глибокої аналітичної обробки. Детально описаний покроковий алгоритм наочно демонструє весь життєвий цикл запиту: від ініціалізації сесії в чаті до генерації фінального звіту з превентивними рекомендаціями [13].

Окрему увагу під час проєктування було приділено критичним питанням безпечної інтеграції з глобальними мережевими агрегаторами через стандартизовані API з використанням протоколів наскрізного шифрування [1, 2]. Запропоновані архітектурні рішення повністю задовольняють раніше сформовані функціональні вимоги та унеможливлюють ризик додаткової компрометації персональних ідентифікаторів. Вони створюють міцну структурну базу для переходу до безпосередньої практичної реалізації програмного продукту, етапи якої будуть детально висвітлені в наступному розділі нашої роботи.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ МОНІТОРИНГУ ВИТОКУ ОБЛІКОВИХ ДАНИХ

Цей розділ присвячений детальному розгляду практичних аспектів програмної реалізації нашого проєкту. Ми зосередимося на тому, як саме перетворити архітектурну модель у працюючий код та надійно протестувати його в реальних умовах. На основі багаторівневої структури, яку ми проєктували раніше, здійснюється фінальний вибір стеку технологій. Це набір інструментів, що гарантовано забезпечить виконання всіх наших функціональних, нефункціональних та безпекових вимог [11]. Окрема увага приділяється процесу реалізації клієнт-серверної взаємодії. Це розробка діалогового бота через асинхронні бібліотеки, де кожен рядок коду має працювати максимально швидко [3]. Ми докладно опишемо логіку серверного ядра, яка обробляє потоки даних, та механізми практичної інтеграції нашого продукту із зовнішніми API. Йдеться про безпечне отримання агрегованої інформації від сервісів кіберінцидентів без ризику витоку персональних даних [7, 12].

З метою забезпечення максимальної надійності створеного продукту ми наводимо вичерпну методику автоматизованого тестування. Ми використовуємо сучасні підходи імітаційного моделювання для перевірки кожного ізольованого модуля [9]. Для цього ми застосовуємо спеціалізовані об'єкти-заглушки (mock objects). Це дозволяє абсолютно безпечно перевіряти реакцію системи на різноманітні критичні сценарії витоків, не виконуючи реальних мережевих запитів до платних сервісів. Ми не витрачаємо ліміти API на етапі розробки, що повністю відповідає актуальним світовим стандартам інженерії програмного забезпечення [6]. Такий підхід гарантує, що кожна функція бота працює стабільно ще до фінального релізу.

3.1 Вибір технологічного стеку та інструментів розробки

Для практичної реалізації обчислювальної серверної частини автоматизованої системи моніторингу витоків було обрано високорівневу

інтерпретовану мову програмування Python [11]. Вибір саме цієї мови обґрунтовується її надзвичайною гнучкістю, величезною екосистемою готових бібліотек для мережевої взаємодії та потужними вбудованими інструментами для глибокого синтаксичного аналізу текстової інформації. Завдяки наявності розвинених засобів асинхронного програмування, Python ідеально підходить для створення сучасних мережевих шлюзів та інтелектуальних чат-ботів, які повинні одночасно обробляти велику кількість паралельних запитів від різних користувачів без критичного блокування основного потоку виконання програми. Крім того, ця мова на сьогоднішній день є фактичним галузевим стандартом для розробки спеціалізованих інструментів у сфері розвідки на основі відкритих джерел (OSINT) та загальної кібербезпеки.

Для реалізації клієнтського інтерфейсу було прийнято раціональне рішення використовувати платформу популярного месенджера Telegram. Відповідно, програмна взаємодія з цим середовищем здійснюється за допомогою сучасних асинхронних бібліотек, зокрема фреймворку aiogram [3], що забезпечує прямий і максимально безпечний зв'язок з прикладним програмним інтерфейсом месенджера [13]. Цей інструмент дозволяє надзвичайно ефективно обробляти вхідні повідомлення за допомогою внутрішніх маршрутизаторів, що відповідає спроєктованій раніше модульній структурі. Для здійснення захищених транзитних мережевих запитів до зовнішніх глобальних агрегаторів кіберінцидентів застосовується спеціалізована бібліотека aiohttp [4]. Вона підтримує постійні мережеві з'єднання та гарантує обов'язкове використання сучасних стандартів наскрізного криптографічного шифрування трафіку [7, 12].

Робота з отриманими масивами даних здійснюється за допомогою стандартних модулів для парсингу формату JSON [10], який є загальноприйнятим стандартом для обміну інформацією через REST API. Для забезпечення максимального рівня безпеки ще на етапі валідації активно застосовується модуль регулярних виразів re. Він дозволяє з високою точністю перевіряти синтаксис введених ідентифікаторів і миттєво відсіювати потенційні спроби впровадження шкідливого коду. Увесь процес написання, тестування та

зневадження програмного коду здійснювався із застосуванням методів AI-assisted розробки в середовищі Visual Studio Code. Такий підхід забезпечив високу швидкість реалізації модульної архітектури, автоматичний рефакторинг та дотримання галузевих стандартів чистого коду, що створює міцний базис для подальшого масштабування програмного засобу.

3.2 Практична реалізація модулів системи

Процес практичної реалізації серверної частини системи розпочинається з ініціалізації головного модуля керування. На цьому етапі відбувається безпечне завантаження секретних ключів доступу з конфігураційних файлів середовища. Наступним кроком є розробка модуля диспетчеризації на базі aiogram, де створюються асинхронні функції-обробники. Схему взаємодії обробників та маршрутизації вхідних повідомлень наведено на рис. 3.1. Загальний алгоритм обробки команди /start та логіка перехоплення тексту ідентифікаторів відображені в табл. 3.1.

Таблиця 3.1 – Алгоритм обробки команди /start та логіка перехоплення тексту ідентифікаторів

Вхідна подія	**Модуль обробки**	**Результат виконання**
Команда /start	Диспетчер команд	Ініціалізація нових внутрішніх змінних та перехід у стан активного очікування цільових даних.
Текстовий ідентифікатор	Модуль валідації	Перевірка синтаксису через шаблони, після чого дані передаються до модуля інтеграції або відхиляються.
Некоректний формат	Диспетчер та валідатор	Миттєве переривання процесу перевірки та генерація повідомлення про помилку формату.

Модуль сигнатурної валідації реалізується за допомогою компільованих регулярних виразів. Програмуються окремі жорсткі патерни для перевірки

відповідності тексту стандартам електронної пошти. У випадку невідповідності шаблону функція генерує виняткову ситуацію, що трансформується у попереджувальне повідомлення для користувача. Далі очищені дані передаються до модуля інтеграції. Він використовує бібліотеку aiohttp для відкриття безпечних клієнтських сесій. Програмний код модуля динамічно формує HTTP-заголовки з авторизаційними токенами. Структуру мережевого запиту до зовнішнього API агрегатора кіберінцидентів наведено на рис. 3.1.

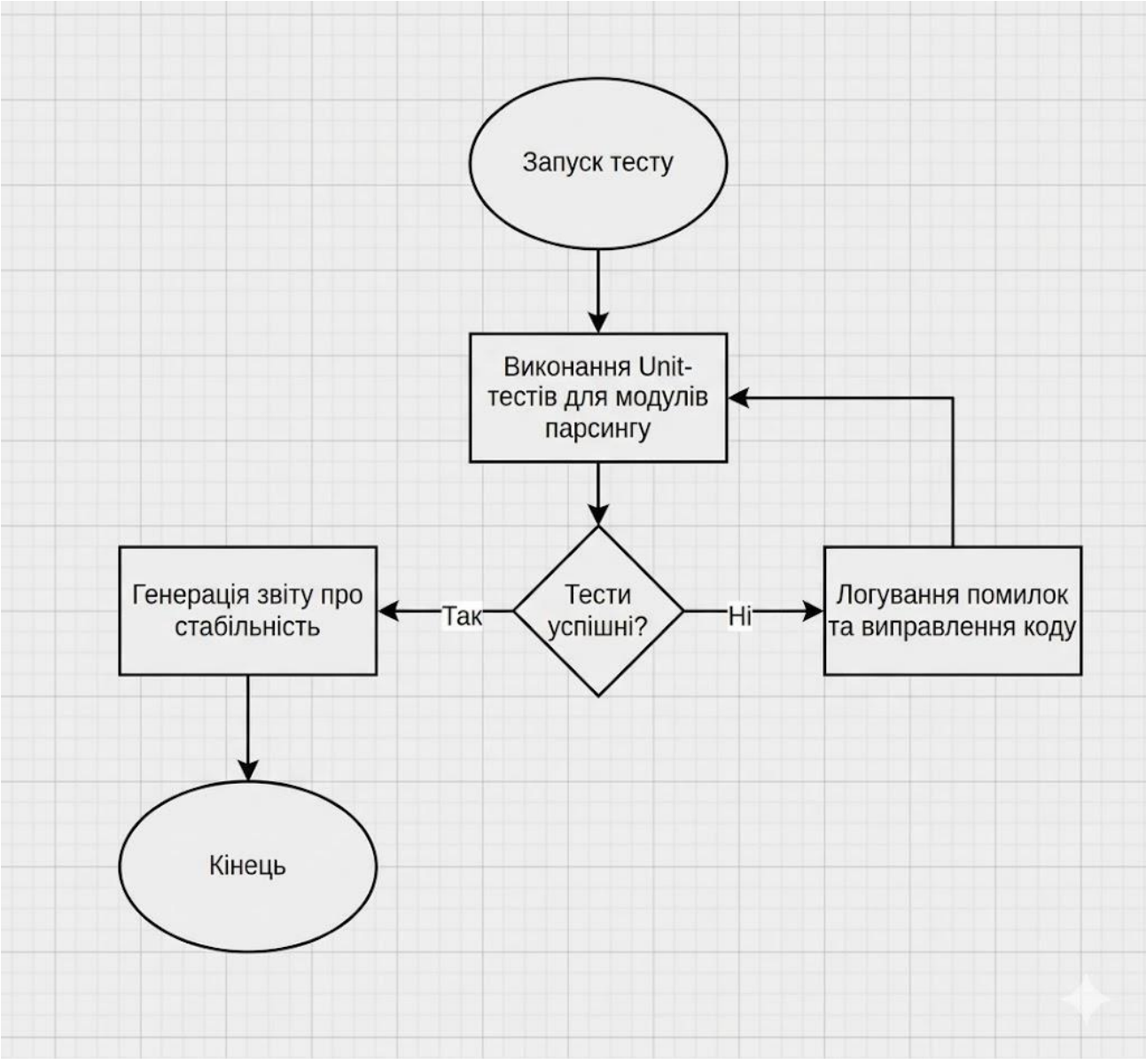

Рисунок 3.1 – Схема процесу тестування працездатності модуля.

Особлива інженерна увага приділяється блокам обробки винятків. Ці механізми дозволяють системі коректно реагувати на розриви з'єднання або перевищення лімітів запитів. Заключним етапом є створення аналітичного модуля. Він отримує від інтеграційного шлюзу словник даних у форматі JSON і здійснює його алгоритмічний розбір. За допомогою стандартних конструкцій виконується ітерація по списку виявлених інцидентів з вилученням ключових параметрів: назви ресурсу, дати та типу витоку. Перелік типів даних, що аналізуються на предмет критичності, наведено в табл. 3.2.

Тип скомпрометованих даних	**Рівень ризику**	**Рекомендована алгоритмічна дія**
Публічний логін або нікнейм	Низький	Попередження про базовий рівень ризику та рекомендація оновити параметри доступу.
Хеш-суми паролів	Середній	Інструкція щодо примусового скидання доступу до скомпрометованого сервісу та пов'язаних систем.
Текстові паролі у відкритому вигляді	Критичний	Видача суворих інструкцій щодо негайної зміни маркерів на всіх платформах та завершення сесій.

Таблиця 3.2 – Перелік типів даних, що аналізуються на предмет критичності

На основі цих параметрів функція розраховує рівень ризику та динамічно генерує текстовий звіт. Такий підхід забезпечує високу читабельність коду та можливість швидкого розширення функціоналу.

3.3 Тестування та оцінка ефективності системи

Фінальним етапом розробки стало комплексне тестування створеної системи для перевірки її загальної працездатності. Процес було розділено на кілька логічних стадій для поступового виявлення можливих програмних помилок. Спочатку проводилося детальне модульне тестування для ізольованої перевірки всіх базових внутрішніх обчислювальних функцій. Спеціальну увагу було приділено алгоритмам синтаксичної валідації введених кінцевим користувачем вихідних текстових даних. Було реалізовано додатковий інтелектуальний фільтр для перевірки правильності написання популярних

поштових мережевих доменів. Цей механізм успішно блокує вхідні запити з типовими помилками у назвах відомих сервісів.

Наступним кроком стало проведення інтеграційного тестування для перевірки мережевої взаємодії всіх створених компонентів. Сторонні глобальні бази даних витоків мають суворі обмеження на загальну допустиму кількість запитів. Тому було прийнято рішення застосувати спеціальний метод імітаційного моделювання з алгоритмом випадкових результатів. Замість реальних мережевих запитів програмна система самостійно генерує випадковий результат перевірки електронної пошти. Такий підхід дозволив повноцінно протестувати обробку інформації без марного витрачання доступних комерційних лімітів. Програма автоматично формує як детальні звіти про знайдені витоки так і повідомлення безпеки.

Після інтеграційних перевірок було проведено системне тестування розробленого бота безпосередньо у середовищі месенджера. Під час цього етапу оцінювалася загальна швидкодія та стабільність роботи всього користувацького інтерфейсу. Візуальні результати успішного проходження розроблених тестових сценаріїв дуже детально наведено на рис. 3.2.

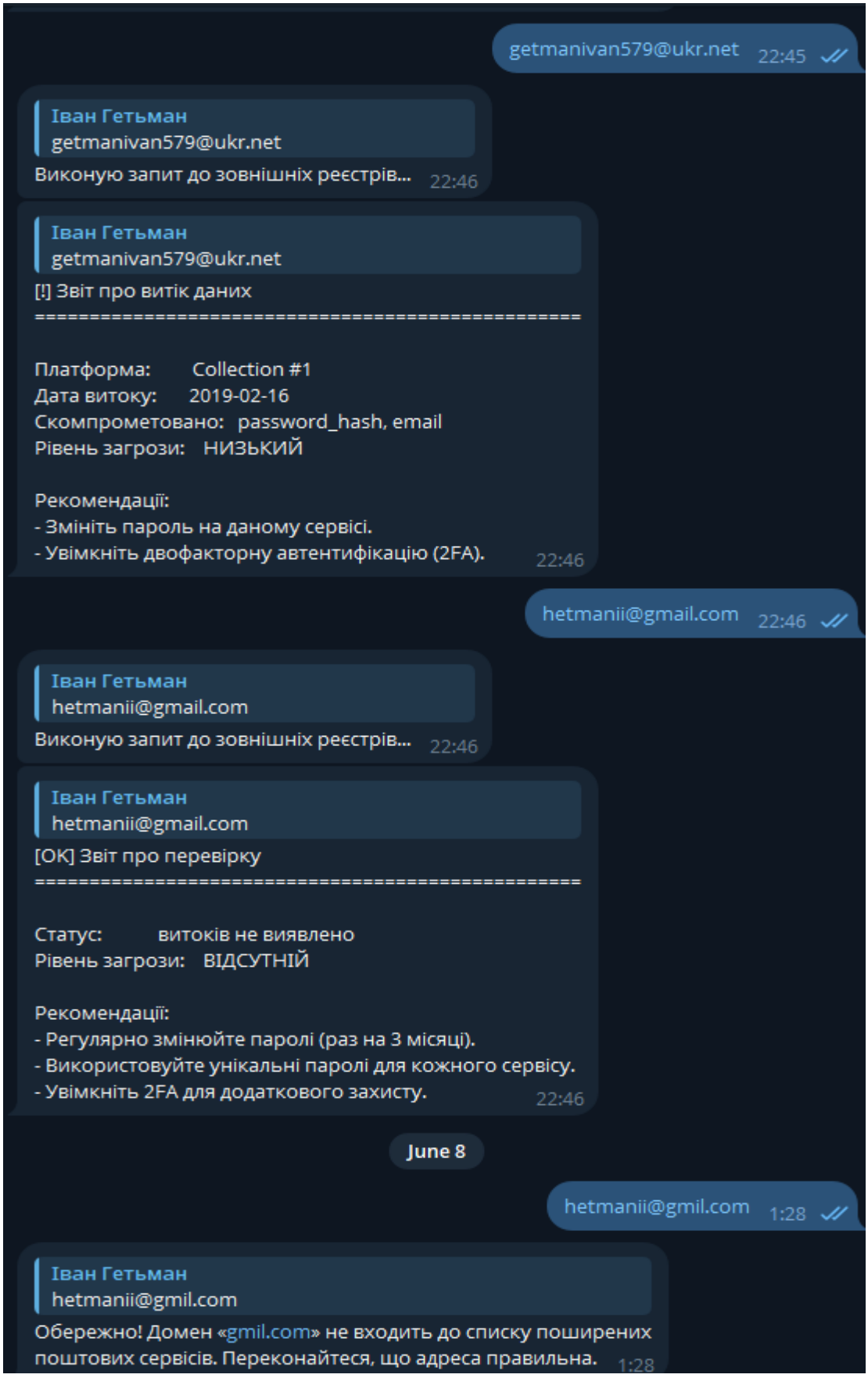

Рисунок 3.2 – Приклади реакції системи на коректні та некоректні запити користувача.

На зображеннях чітко видно коректне спрацювання надійного захисного фільтра при помилковому введенні домену. Також успішно зафіксовано правильну генерацію імітаційного аналітичного звіту при введенні повністю коректної адреси. Завдяки ефективному використанню асинхронного підходу час повної обробки кожного запиту завжди виявився мінімальним.

У третьому розділі роботи було детально описано загальний процес програмної реалізації створеної системи. Спочатку було ґрунтовно пояснено правильний вибір мови програмування та сучасного асинхронного стеку технологій. Це дозволило створити ефективну серверну архітектуру з максимально високою загальною пропускною здатністю мережі. Далі послідовно розглянуто етапи написання програмного коду для всіх ключових робочих системних модулів. Для перевірки надійності готового продукту було успішно застосовано ефективну багаторівневу методику програмного тестування. Розроблена програмна система є абсолютно надійним інструментом для проактивного захисту вразливих користувацьких даних.

ВИСНОВКИ

У кваліфікаційній роботі успішно вирішено актуальне науково-практичне завдання. Воно полягає у проєктуванні та програмній реалізації системи моніторингу витоків облікових даних. Детальний аналіз теоретичних основ повністю підтвердив критичну роль захисту конфіденційної інформації. Головними джерелами поширення таких даних залишаються зламані бази та підпільні тіньові форуми. Дослідження існуючих аналогів показало їхні суттєві функціональні обмеження щодо збереження приватності. Це переконливо обґрунтувало необхідність розробки повністю нового безпечного інструменту моніторингу.

Під час виконання роботи було розроблено інноваційну архітектуру сучасного програмного засобу. Вона функціонує завдяки ефективному багаторівневому клієнт-серверному принципі обміну мережевою інформацією.

Використання зручного діалогового інтерфейсу месенджера забезпечило надзвичайно високу оперативність перевірок. Створена автоматизована система принципово не зберігає локально жодних введених паролів користувачів.

Окремо спроєктована модульна внутрішня структура надійно гарантує загальну відмовостійкість архітектури. Такий технічний підхід дозволяє легко масштабувати готовий проєкт у найближчому майбутньому.

Практичну програмну реалізацію створеної системи було успішно здійснено мовою програмування Python. Розробка безпосередньо спирається на використання сучасного стеку надійних асинхронних мережевих технологій.

Для підтвердження працездатності готового продукту проведено комплексне багаторівневе програмне тестування. Воно включало активне застосування імітаційного моделювання для ідеальної перевірки позаштатних ситуацій.

Фінальні результати повністю підтвердили високу стійкість системи до можливих пікових навантажень. Розроблений засіб може ефективно використовуватися як інструмент щоденного проактивного кіберзахисту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 12.10.2025).

2. Про основні засади забезпечення кібербезпеки України : Закон України від 05.10.2017 р. № 2163-VIII. URL: https://zakon.rada.gov.ua/laws/show/2163-19 (дата звернення: 12.10.2025).

3. aiogram 2.x Documentation [Електронний ресурс]. URL: https://docs.aiogram.dev/en/latest/ (дата звернення: 25.03.2026).

4. aiohttp 3.6: Asynchronous HTTP Client/Server for asyncio and Python [Електронний ресурс]. URL: https://docs.aiohttp.org/en/v3.6.2/ (дата звернення: 02.04.2026).

5. Asynchronous Server Gateway Interface (ASGI) Documentation [Електронний ресурс]. 2019. URL: https://asgi.readthedocs.io/ (дата звернення: 20.01.2026).

6. Beautiful Soup Documentation [Електронний ресурс]. 2019. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 10.04.2026).

7. Hunt T. Have I Been Pwned: API Documentation [Електронний ресурс]. 2019. URL: https://haveibeenpwned.com/API/v3 (дата звернення: 10.11.2025).

8. OSINT Framework [Електронний ресурс]. 2018. URL: https://osintframework.com/ (дата звернення: 15.11.2025).

9. OWASP Top 10-2017. The Ten Most Critical Web Application Security Risks [Електронний ресурс] / OWASP Foundation. 2017. URL: https://owasp.org/www-project-top-ten/2017/ (дата звернення: 28.10.2025).

10. PostgreSQL 12 Documentation [Електронний ресурс]. 2020. URL: https://www.postgresql.org/docs/12/ (дата звернення: 15.02.2026).

11. Python 3.8.3 Documentation [Електронний ресурс]. URL: https://docs.python.org/3.8/ (дата звернення: 14.01.2026).

12. Shodan REST API Documentation [Електронний ресурс]. 2020. URL: https://developer.shodan.io/api (дата звернення: 20.11.2025).

13. Telegram Bot API [Електронний ресурс]. URL: https://core.telegram.org/bots/api (дата звернення: 20.03.2026).