НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВИКОРИСТАННЯ РЕСУРСОМІСТКИХ ПАСТОК ДЛЯ УСКЛАДНЕННЯ АТАКИ НА ВЕБСЕРВЕР»

Виконав: здобувач 4 курсу
Рівень бакалавр
Галузь знань 12 «Інформаційні технології»,
спеціальність 125 «Кібербезпека»
Спесівцев Микола Андрійович

Керівник: к.т.н., доцент
Бойко Віктор Дмитрович

Рецензент: к.т.н., доцент
Фразе-Фразенко Олексій Олексійович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

Факультет кібербезпеки та інформаційних технологій
Кафедра **кібербезпеки**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **125 Кібербезпека**
Назва освітньої програми: **Кібербезпека**

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
професор С.А. Горбаченко
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Спесівцеву Миколі Андрійовичу

1. Тема роботи: «Використання ресурсомістких пасток для ускладнення атаки на вебсервер»
Керівник роботи: к.т.н., доцент Віктор Дмитрович Бойко
затверджені наказом НУ ОЮА від «18» березня 2025 року № 548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання«16 » вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та пошук науково-технічної інформації за темою роботи	Вересень-жовтень 2024	
2	Узгодження теми з науковим керівником, формулювання мети завдань дослідження	Листопад 2024	
3	Робота над першим розділом	Листопад-грудень 2024	
4	Робота над другим розділом	Січень-лютий 2025	
5	Робота над третім розділом	Лютий-березень 2025	
6	Уточнення подальшого обсягу роботи та його коригування	Березень-травень 2025	
7	Оформлення звітної частини	Травень 2025	
8	Захист роботи	12.06.2025	

Здобувач_____
(підпис) (прізвище та ініціали)

Керівник роботи_____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Спесівцев Микола Андрійович "Використання ресурсомістких пасток для ускладнення атаки на вебсервер" – Кваліфікаційна робота бакалавра за спеціальністю 125 "Кібербезпека", освітньою програмою "Кібербезпека". Кваліфікаційна робота викладена на 42 сторінках основного тексту і сторінках загального тексту, містить 3 розділи, 22 рисунків, 3 додатки та 14 використаних джерел.

Метою дослідження є розробка та аналіз ефективності програмного засобу типу "ресурсомістка пастка" (HTTP Tarpit) для уповільнення автоматизованих атак на вебсервери, збору детальної інформації про атакуючих ботів та інтеграції з сервісами репутації IP-адрес для підвищення загального рівня безпеки.

У кваліфікаційній роботі реалізовано механізм уповільненої відповіді на HTTP-запити, систему збору та структурування даних про атаки (IP, заголовки, шляхи, User-Agent) у базу даних SQLite, інтеграцію з базами GeoIP для геолокації та визначення ASN атакуючих, а також автоматичне надсилання звітів про шкідливі IP-адреси до сервісу AbuseIPDB. Проведено тестування розробленого програмного засобу та підготовлено основу для подальшого аналізу зібраних даних.

Розроблений HTTP тарпіт може бути використаний як додатковий засіб захисту для вебсерверів різного масштабу, дозволяючи знизити ефективність дій автоматизованих систем сканування та атаки, а також зібрати цінну інформацію для проактивного блокування загроз.

Можливими напрямками подальших досліджень є вдосконалення алгоритмів уповільнення, розширення механізмів аналізу зібраних даних, а також інтеграція з іншими системами безпеки.

ВЕБ СЕРВЕР, ТАРПІТ, HTTP, БОТ, ВЕБ-БЕЗПЕКА, PYTHON, AIOHTTP, ASYNCIO, ABUSEIPDB, GEOIP, SQLITE, АНАЛІЗ ДАНИХ.

ABSTRACT

Spesivtsev Mykola Andreevich “The use of resource-intensive traps to complicate an attack on a web server” – Bachelor`s qualification work in the specialty 125 "Cybersecurity", educational program "Cybersecurity". The qualification work is presented on 42 pages of the main text and pages of the general text, contains 3 sections, 22 figures, 3 appendices and 14 sources used.

The purpose of the study is to develop and analyze the effectiveness of a "resource-intensive trap" software tool (HTTP Tarpit) to slow down automated attacks on web servers, collect detailed information about attacking bots, and integrate with IP reputation services to improve the overall level of security.

The qualification work implemented a mechanism for slow response to HTTP requests, a system for collecting and structuring data on attacks (IP, headers, paths, User-Agent) to the SQLite database, integration with GeoIP databases for geolocation and determining the ASN of attackers, as well as automatic sending of reports on malicious IP addresses to the AbuseIPDB service. The developed software tool was tested and the basis for further analysis of the collected data was prepared.

The developed HTTP tarpit can be used as an additional means of protection for web servers of various sizes, allowing you to reduce the effectiveness of automated scanning and attack systems, as well as collect valuable information for proactive blocking of threats.

Possible areas of further research are the improvement of deceleration algorithms, the expansion of mechanisms for analyzing the collected data, as well as integration with other security systems.

WEB SERVER, TARPIT, HTTP, BOT, WEB SECURITY, PYTHON, AIOHTTP, ASYNCIO, ABUSEIPDB, GEOIP, SQLITE, DATA ANALYSIS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Прикладний Програмний Інтерфейс (Application Programming Interface)

ASN – Номер Автономної Системи (Autonomous System Number)

БД – База Даних

СКБД – Система Керування Базами Даних

HTTP – Протокол Передачі Гіпертексту (HyperText Transfer Protocol)

HTTPS – Захищений Протокол Передачі Гіпертексту (HyperText Transfer Protocol Secure)

IP – Інтернет-Протокол (Internet Protocol)

URL – Уніфікований Покажчик Ресурсу (Uniform Resource Locator)

DNS – Система Доменних Імен (Domain Name System)

SSL – Протокол Захищених Сокетів (Secure Sockets Layer)

TLS – Протокол Захисту Транспортного Рівня (Transport Layer Security)

WAF – Міжмережевий Екран Веб-додатків (Web Application Firewall)

IDS – Система Виявлення Вторгнень (Intrusion Detection System)

IPS – Система Запобігання Вторгненням (Intrusion Prevention System)

DDoS – Розподілена Атака типу "Відмова в Обслуговуванні" (Distributed Denial of Service)

UA (User-Agent) – Агент Користувача

VPS – Віртуальний Приватний Сервер (Virtual Private Server)

SIEM – Система Управління Інформаційною Безпекою та Подіями (Security Information and Event Management)

TCP – Протокол Керування Передачею (Transmission Control Protocol)

ВСТУП

Сучасний цифровий світ характеризується невпинним зростанням обсягів інформації та кількості сервісів, доступних через мережу Інтернет. Вебсервери стали невід`ємною частиною інфраструктури будь-якої організації, забезпечуючи доступ до вебсайтів, онлайн-додатків, API та інших важливих ресурсів. Водночас, ця доступність робить їх привабливою мішенню для різноманітних кібератак. Однією з найпоширеніших загроз є активність шкідливих автоматизованих програм, відомих як боти, та цілих бот-мереж, які використовуються для широкого спектра зловмисних дій. Такі дії включають сканування на наявність вразливостей, атаки типу "відмова в обслуговуванні" (далі - DDoS), зокрема HTTP Flood, спроби підбору облікових даних (брутфорс), розповсюдження спаму, автоматизований збір контенту (скрейпінг) та іншу деструктивну діяльність. Наслідки таких атак можуть бути вкрай негативними: від фінансових збитків та простою сервісів до репутаційних втрат та компрометації конфіденційних даних.

Традиційні засоби захисту вебсерверів, такі як міжмережеві екрани (Firewalls) та системи виявлення/запобігання вторгненням (далі - IDS/IPS), а також спеціалізовані міжмережеві екрани веб-додатків (далі - WAF), відіграють важливу роль у забезпеченні безпеки. Однак, вони не завжди є достатньо ефективними проти сучасних, адаптивних ботів, які вміють обходити прості сигнатурні методи виявлення або імітувати поведінку легітимних користувачів[1]. Крім того, вартість та складність налаштування деяких комерційних рішень можуть бути значним бар`єром, особливо для малих та середніх підприємств або некомерційних проектів.

У цьому контексті актуальним напрямком досліджень та розробок є створення та застосування засобів активної оборони, які не просто блокують підозрілий трафік, а взаємодіють з атакуючим з метою його уповільнення, введення в оману та збору цінної інформації. Одним із таких підходів є використання ресурсомістких пасток, або тарпітів (tarpits). Принцип дії тарпіта полягає у навмисному уповільненні мережевих

з`єднань або відповідей на запити, що змушує атакуючого бота витрачати значно більше власних ресурсів (час, обчислювальна потужність, мережеві з`єднання) на взаємодію з пасткою, ніж з реальним сервісом. Це не тільки знижує ефективність атаки, але й дозволяє зібрати детальну інформацію про атакуючого для подальшого аналізу та блокування. Таким чином, розробка ефективних HTTP тарпітів є актуальною задачею, спрямованою на підвищення стійкості вебсерверів до автоматизованих загроз.

Об`єктом дослідження є ресурсомістка пастка (далі - HTTP Tarpit) як програмний засіб для ускладнення автоматизованих атак на вебсервер.

Предметом дослідження є процеси взаємодії шкідливих ботів з вебсервером та методи протидії їм шляхом уповільнення мережевих протоколів на прикладному рівні, а також механізми збору, зберігання та використання інформації про атакуючих ботів.

Метою даної дипломної роботи є розробка та дослідження програмного засобу типу HTTP тарпіт, здатного ефективно уповільнювати автоматизовані атаки, збирати розширену інформацію про атакуючих, включаючи геолокаційні дані та дані про автономні системи, та автоматично звітувати про шкідливі IP-адреси до сервісу AbuseIPDB.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести аналіз сучасних автоматизованих загроз для вебсерверів та існуючих методів протидії, з акцентом на ресурсомісткі пастки.
2. Обґрунтувати вибір архітектури та технологічного стеку для розробки HTTP тарпіта, враховуючи вимоги до продуктивності та надійності.
3. Розробити програмний модуль HTTP тарпіта з механізмом уповільненої відповіді на базі асинхронного програмування.
4. Реалізувати систему збору та структурування даних про атаки, включаючи IP-адресу, HTTP-заголовки, запитувані шляхи, User-Agent.
5. Інтегрувати механізми отримання геолокаційної інформації (країна, місто) та

даних про ASN для кожного IP-адреси атакуючого.

6. Розробити систему зберігання зібраних даних у локальній базі даних SQLite.

7. Реалізувати модуль для автоматичної взаємодії з API сервісу AbuseIPDB для надсилання звітів про шкідливі IP.

8. Підготувати розроблений програмний засіб до розгортання на сервері з використанням Nginx як зворотного проксі та налаштуванням HTTPS.

Методи дослідження, що використовуються в роботі, включають аналіз науково-технічної літератури, системний аналіз, об`єктно-орієнтоване проектування та програмування, експериментальне тестування розробленого програмного засобу, а також методи статистичного аналізу даних.

1 АНАЛІЗ ПРОБЛЕМИ ЗАХИСТУ ВЕБСЕРВЕРІВ ТА РОЛІ РЕСУРСОМІСТКИХ ПАСТОК

1.1 Сучасні автоматизовані загрози для вебсерверів та актуальність їхнього ускладнення

Вебсервери, що є фундаментальною основою функціонування глобальної мережі Інтернет та численних цифрових сервісів, незмінно перебувають під прицілом кіберзлочинців. Особливу стурбованість викликає ескалація та ускладнення автоматизованих загроз, які реалізуються за допомогою спеціалізованого програмного забезпечення – ботів, та їх координованих об`єднань – бот-мереж, або заражених шкідливим ПЗ ботнетів (рисунок 1.1).

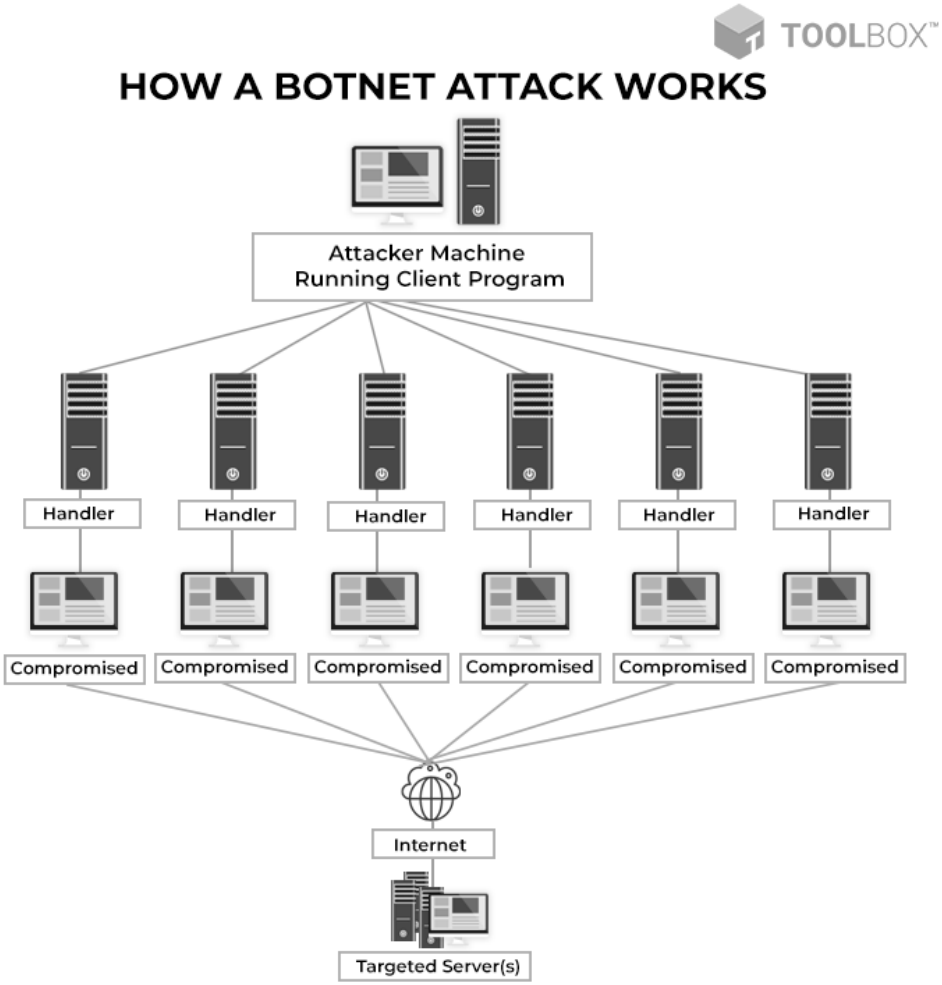


Рисунок 1.1 – Принцип роботи ботнетів [2]

Ці автоматизовані системи характеризуються здатністю генерувати надзвичайно високу інтенсивність запитів, що дозволяє їм здійснювати як масові розвідувальні операції, так і руйнівні атаки. За оцінками провідних аналітичних центрів у сфері кібербезпеки, бот-трафік становить значну, а іноді й переважну частку всього інтернет-трафіку, причому значний відсоток цього трафіку класифікується як шкідливий або небажаний. Особливої гостроти ця проблема набула в Україні в умовах повномасштабної гібридної війни, коли ворожі кіберпідрозділи активно використовують бот-мережі для атак на державні інформаційні ресурси, об`єкти критичної інфраструктури та медіа.

Сучасні автоматизовані загрози демонструють високий рівень адаптивності та витонченості. Вони активно використовують розподілені інфраструктури, що ускладнює їх блокування за статичними ознаками, такими як IP-адреси, а також застосовують техніки маскування під легітимний трафік, імітуючи поведінку реальних користувачів та їхніх програмних агентів. Серед найбільш поширених та небезпечних проявів такої активності можна виділити систематичне сканування на наявність вразливостей. Боти цього типу безперервно досліджують веб-додатки на предмет відомих слабкостей конфігурації, застарілих версій програмного забезпечення, а також специфічних вразливостей прикладного рівня, таких як SQL-ін`єкції, міжсайтовий скриптинг чи помилки, що дозволяють несанкціоноване включення файлів. Не менш розповсюдженими є атаки Brute-force, або так звані “атаки грубої сили” – вид зламу, при якому хакер або тестувальник методом перебору намагається вгадати дані для входу в систему**.** Класичний приклад – коли намагаються підібрати правильне ім’я користувача чи пароль шляхом перебору сотень комбінацій. Наприклад, типовою ціллю таких атак можуть бути адміністративні панелі популярних в Україні систем управління контентом, таких як WordPress або Joomla, що використовуються багатьма новинними порталами чи сайтами громадських організацій [3].

Критичною загрозою залишаються розподілені атаки на відмову в

обслуговуванні (DDoS), особливо атаки на рівні додатку (L7 DDoS), коли генерується масовий потік легітимних на вигляд HTTP-запитів до ресурсомістких компонентів веб-додатку. Такі дії спрямовані на вичерпання обчислювальних ресурсів сервера, його пам`яті або пропускної здатності мережевого каналу, що призводить до повної або часткової недоступності сервісу для легітимних користувачів [4].

Актуальність розробки ефективних методів ускладнення діяльності таких автоматизованих систем визначається обмеженнями традиційних засобів захисту. Сигнатурні методи виявлення, що лежать в основі багатьох міжмережевих екранів веб-додатків (WAF) та систем виявлення/запобігання вторгненням (IDS/IPS), часто виявляються неефективними проти нових або модифікованих ботів. Динамічний характер сучасних бот-мереж, що використовують великі пули IP-адрес та проксі-серверів, робить простий механізм блокування за IP малоефективними [5]. Навіть такі методи, як CAPTCHA, все частіше успішно долаються автоматизованими системами.

У відповідь на ці виклики, перспективним напрямком є розвиток технологій активної оборони та обману, серед яких особливе місце займають ресурсомісткі пастки (тарпіти). На відміну від стратегій негайного блокування, тарпіти цілеспрямовано уповільнюють взаємодію з підозрілими клієнтами, змушуючи автоматизовані системи атакуючого витрачати значно більше власних ресурсів на кожен запит. Такий підхід не лише знижує загальну ефективність масових автоматизованих атак, але й надає захисникам час на виявлення, аналіз та реагування. Крім того, інформація, зібрана під час взаємодії атакуючого з тарпітом, є надзвичайно важливою для профілювання загроз. Таким чином, дослідження та розробка ефективних HTTP тарпітів є важливим і актуальним завданням у контексті забезпечення стійкості сучасних веб-інфраструктур.

1.2 Огляд існуючих методів та засобів протидії бот-атакам

Ефективна протидія автоматизованим атакам на вебсервери вимагає

комплексного підходу, що поєднує різноманітні технології та стратегії. Існуючі методи можна умовно розділити на превентивні, спрямовані на запобігання атакам, та реактивні, що включають виявлення та активну протидію вже розпочатим інцидентам. Кожен з цих підходів має свої переваги та обмеження, що визначають доцільність їх застосування в конкретних сценаріях.

Превентивні методи захисту традиційно включають використання міжмережевих екранів та систем виявлення/запобігання вторгненням . Мережеві файрволи працюють на транспортному та мережевому рівнях моделі OSI, фільтруючи трафік на основі IP-адрес, портів та протоколів. Хоча вони є необхідним елементом захисту периметра, їхня ефективність проти складних атак на рівні додатку (L7), які часто здійснюють боти, є обмеженою. Системи IDS/IPS аналізують мережевий трафік або лог-файли на наявність відомих сигнатур атак або аномальної поведінки. За даними деяких досліджень, до 70% успішних вторгнень могли б бути попереджені за допомогою правильно налаштованих IDS/IPS, однак сигнатурний аналіз залишається вразливим до нових атак ("zero-day"), а евристичні методи можуть генерувати значну кількість хибних спрацювань (false positives), що, за оцінками, може сягати 10-15% від усіх сповіщень у деяких конфігураціях [6].

Більш спеціалізованим засобом захисту є WAF. WAF функціонують на прикладному рівні, аналізуючи HTTP/HTTPS трафік та застосовуючи набори правил для виявлення та блокування атак . Сучасні WAF часто включають модулі для протидії ботам. Дослідження ринку показують, що компанії, які впровадили WAF, відзначають зниження успішних атак на веб-додатки в середньому на 50-75%. Перевагою WAF є їхня здатність забезпечувати глибокий аналіз трафіку. Недоліками є потенційна складність налаштування, можливість обходу деякими витонченими ботами.

Для відрізнення людей від автоматизованих програм широко використовуються CAPTCHA та її аналоги. Хоча початкова ефективність була високою, розвиток алгоритмів машинного навчання призвів до того, что сучасні системи розпізнавання

можуть долати більшість типів CAPTCHA з успішністю понад 90% у деяких випадках [7]. Це, разом з негативним впливом на досвід користувача, знижує привабливість цього методу як основного.

Механізми обмеження частоти запитів (Rate Limiting) дозволяють контролювати кількість запитів від одного клієнта. Це ефективно проти простих брутфорс-атак, знижуючи їх швидкість у десятки разів. Однак, боти, що є частиною розподілених бот-мереж, які можуть налічувати від тисяч до мільйонів унікальних IP-адрес, легко обходять такі обмеження.

Аналіз репутації IP-адрес та геолокаційне блокування базуються на використанні чорних списків та блокуванні трафіку з певних географічних регіонів. Застосування актуальних чорних списків може допомогти заблокувати до 30-40% відомого шкідливого трафіку. Недоліком є те, що зловмисники постійно змінюють IP-адреси, а легітимні користувачі можуть випадково потрапити під блокування.

Реактивні методи та засоби активної оборони включають ханіпоти (honeypots) та ресурсомісткі пастки (tarpits). Ханіпоти імітують реальні системи для приваблення зловмисників та вивчення їхніх дій. Дослідження показують, що правильно налаштований ханіпот може зібрати інформацію про десятки тисяч атак протягом місяця, надаючи цінні дані про нові вектори та інструменти.

Ресурсомісткі пастки (tarpits), на відміну від ханіпотів, фокусуються на максимальному уповільненні взаємодії з атакуючим. Експерименти з HTTP тарпітами показують, що час утримання одного з`єднання з ботом може бути збільшено з мілісекунд до десятків секунд або навіть хвилин, що при масовій атаці призводить до значного вичерпання ресурсів атакуючої сторони. Головна перевага тарпітів полягає у низькому споживанні ресурсів захисника порівняно з потенційними витратами атакуючого. До недоліків можна віднести те, що боти з агресивними таймаутами можуть швидко розривати з`єднання. Однак, поєднання уповільнення з можливістю збору даних робить HTTP тарпіти перспективним інструментом для доповнення існуючих систем захисту.

1.3 Концепція та принципи функціонування ресурсомістких пасток (тарпітів) як засобу активної оборони

Історично, однією з перших відомих реалізацій тарпіта була система LaBrea, розроблена Томом Лістерманом на початку 2000-х років. LaBrea працювала на мережевому рівні, відповідаючи на ARP-запити для невикористовуваних IP-адрес у локальній мережі, а потім надзвичайно повільно обробляла SYN-пакети, що надходили на ці IP-адреси. Це призводило до того, що сканери портів або хробаки, які намагалися підключитися до цих неіснуючих хостів, "зависали" на тривалий час, очікуючи завершення TCP-рукостискання, тим самим вичерпуючи свої таблиці станів з`єднань та уповільнюючи поширення [8] (рисунок 1.2).

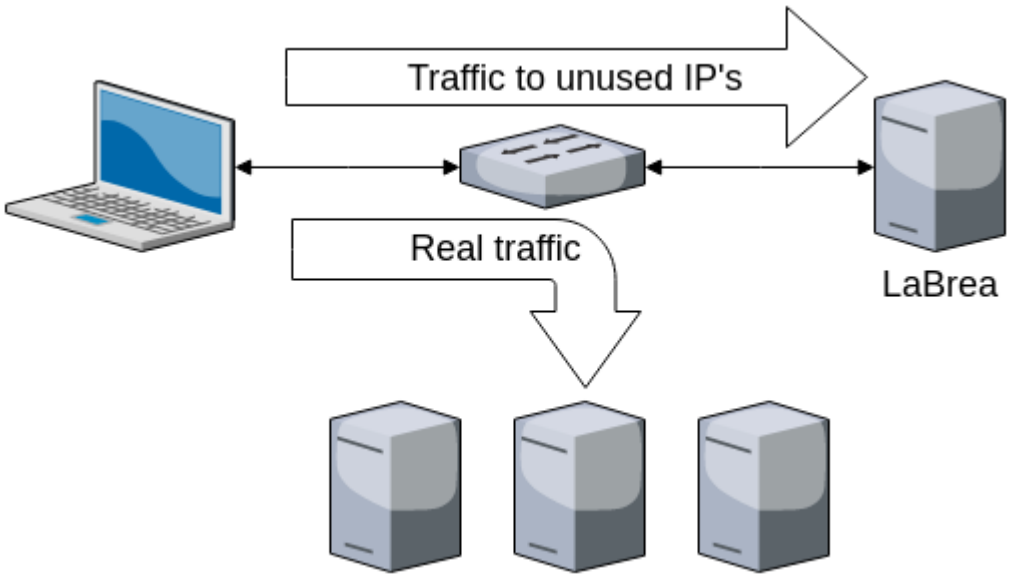


Рисунок 1.2 – Принцип роботи LaBrea [8]

Основний принцип функціонування тарпіта полягає у створенні асиметрії у витратах ресурсів між атакуючим та захисником. Тарпіт розробляється таким чином, щоб вимагати мінімальних ресурсів від захисної системи (пам`ять, процесорний час), але при цьому змушувати клієнта (потенційного зловмисника) утримувати з`єднання та очікувати відповіді якомога довше. Це досягається шляхом маніпуляцій на різних рівнях мережевої взаємодії:

Тарпіти на рівні TCP: Як і LaBrea, ці системи можуть уповільнювати процес встановлення TCP-з`єднання (наприклад, затримуючи відправку SYN-ACK пакетів) або маніпулювати параметрами TCP-сесії після її встановлення (наприклад, анонсуючи нульовий розмір вікна TCP (Zero Window), що змушує відправника призупинити передачу даних, або відправляючи дані дуже малими сегментами з великими паузами) [8].

Linux Netfilter TARPIT, входить до пакету xtables-addons

LaBrea – використовує вільні IP адреси для логування та уповільнення атакуючих

Honeyd – Інструмент виявлення атак та несанкціонованої активності. Використовує вільні IP адреси, емулює різні ОС та послуги

Mikrotik Router OS – Дозволяє налаштувати довільну логіку захисту хостів та підмереж.

Швидше за все, схожа функціональність є у Cisco, Juniper, Huawei, Fortinet Fortigate, Imperva, pfSense, NG Firewall, etc.

Тарпіти на прикладному рівні (Application-level Tarpits): Ці тарпіти імітують поведінку певного сервісу (HTTP, SSH, SMTP тощо), але роблять це навмисно повільно. Наприклад, HTTP тарпіт може швидко надіслати клієнту HTTP-заголовки відповіді (наприклад, 200 OK), створюючи ілюзію нормальної роботи сервера, а потім почати передавати тіло відповіді по одному байту з інтервалом у кілька секунд. Саме такий підхід є предметом даної дипломної роботи. Перевагою тарпітів прикладного рівня є можливість збору більш детальної інформації про запити атакуючого (URL, заголовки, методи) та менша ймовірність впливу на легітимний трафік, якщо тарпіт реалізовано як окремий сервіс. Прикладом тарпіта для SSH є endlessh, який надсилає нескінченний потік випадкових даних після встановлення SSH-з`єднання, не даючи клієнту дійти до етапу автентифікації [9].

Endlessh – емуляція SSH сервісу, який змушує клієнта нескінченно тримати з`єднання.

Portspoof – вміє прикидатися безліччю сервісів, віддає їх банери.

SMTP – затримка відповіді на вхідні SMTP команди, застосовується для боротьби з розсилкою спаму, є в багатьох продуктах (наприклад CommuniGate).

Для наочності давайте уявимо ситуацію, де ми маємо сервер, на якому запущена служба RDP, а міжмережевого екрану немає (рисунок 1.3).

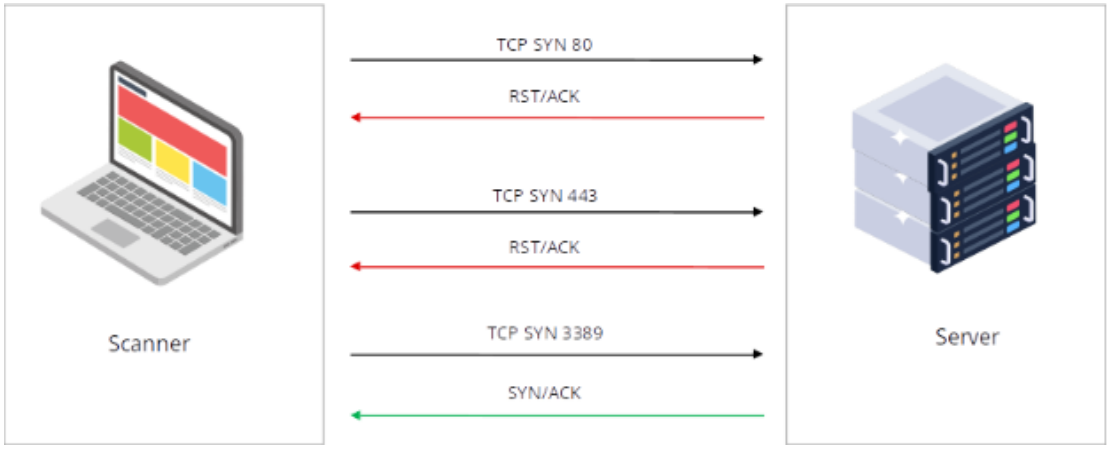


Рисунок 1.3 – Відношення клієнт (сканер) – сервер при вимкненому WAF

З рисунку 1.3. видно, що атакуючий має чіткий сигнал про доступність порту. Сканування займе найменшу кількість часу.

Тепер увімкнемо міжмережевий екран і застосуємо політику DROP (рисунок 1.4).

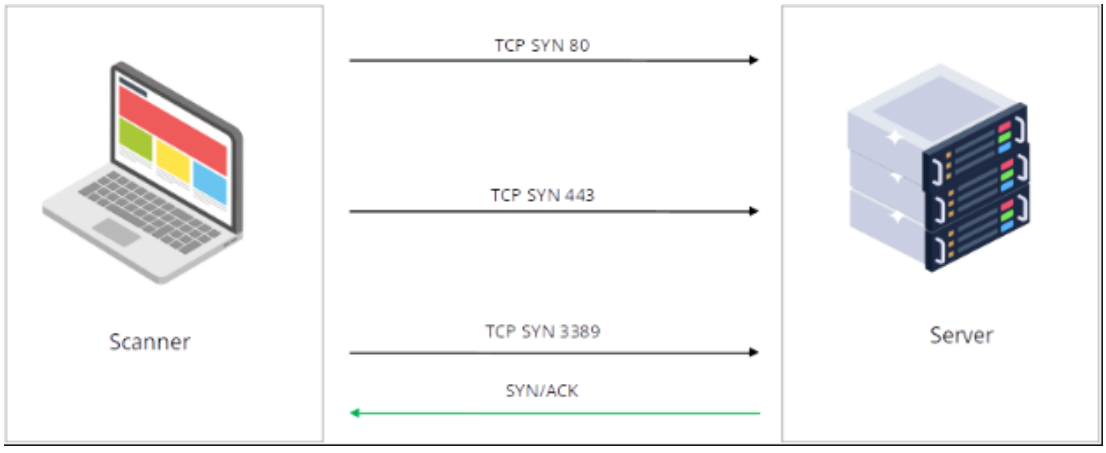


Рисунок 1.4 – Відношення клієнт (сканер) – сервер при увімкненому WAF

Сканування займе більше часу, тому що сканер чекатиме закінчення таймауту на SYN, але все одно дізнатися які порти відкриті не проблема (рисунок 1.5).

Тепер увімкнемо tarpit.

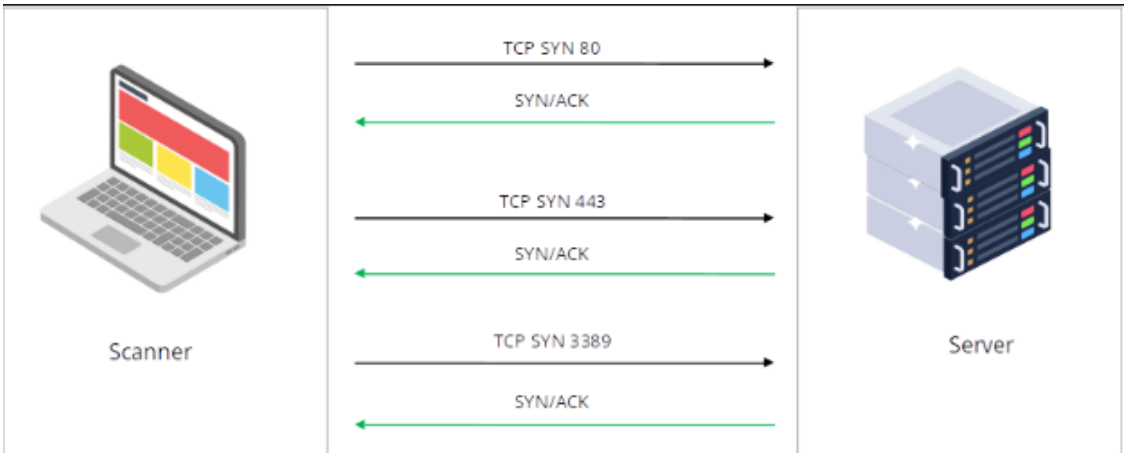


Рисунок 1.5– Відношення клієнт (сканер) – сервер при увімкненому tarpit

Всі порти «відкриті», але передавати дані по них не вийде, сканування такого хоста займе багато часу, але корисної інформації не буде отримано.

2 ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ HTTP ТАРПІТА

2.1 Визначення вимог та обґрунтування вибору архітектурних рішень і технологій

Розробка ефективного програмного засобу типу HTTP тарпіт, призначеного для ускладнення автоматизованих атак та збору інформації про зловмисників, потребує ретельного визначення комплексу вимог та обґрунтованого вибору технологічного стеку й архітектурних підходів. Ключовими функціональними вимогами до системи є здатність приймати та обробляти HTTP/HTTPS з`єднання на стандартних веб-портах, реалізація механізму уповільненої відповіді для вичерпання ресурсів атакуючого, здійснення детального збору інформації про клієнта та запит (IP-адреса, HTTP-заголовки, шлях, User-Agent), збагачення зібраних даних геолокаційною інформацією (країна, місто, ASN) за допомогою локальних баз GeoIP, зберігання даних у структурованій реляційній базі даних SQLite, автоматичне надсилання звітів про шкідливі IP-адреси до сервісу AbuseIPDB, а також забезпечення можливості конфігурації параметрів роботи та коректного визначення реальної IP-адреси клієнта при роботі за зворотним проксі-сервером. Нефункціональні вимоги включають забезпечення високої продуктивності при обробці великої кількості одночасних повільних з`єднань, стабільність роботи, безпеку самого тарпіта та простоту його розгортання й адміністрування.

Враховуючи специфіку завдання – необхідність ефективно обробляти велику кількість одночасних мережевих з`єднань, які за своєю природою є I/O-bound (операції вводу-виводу, що очікують на мережеву відповідь), – вибір мови програмування та моделі виконання є критично важливим архітектурним рішенням. Перевага надається технологіям, що підтримують асинхронне програмування (рисунок 2.1) на основі циклу подій (event loop), оскільки такий підхід дозволяє одному потоку виконання обслуговувати безліч з`єднань без блокування на операціях

очікування. Для реалізації програмного засобу було обрано мову програмування Python. Цей вибір обґрунтовується високою швидкістю розробки, читабельністю коду, потужною вбудованою підтримкою асинхронності через модуль asyncio та ключові слова async/await, а також наявністю зрілих асинхронних бібліотек.

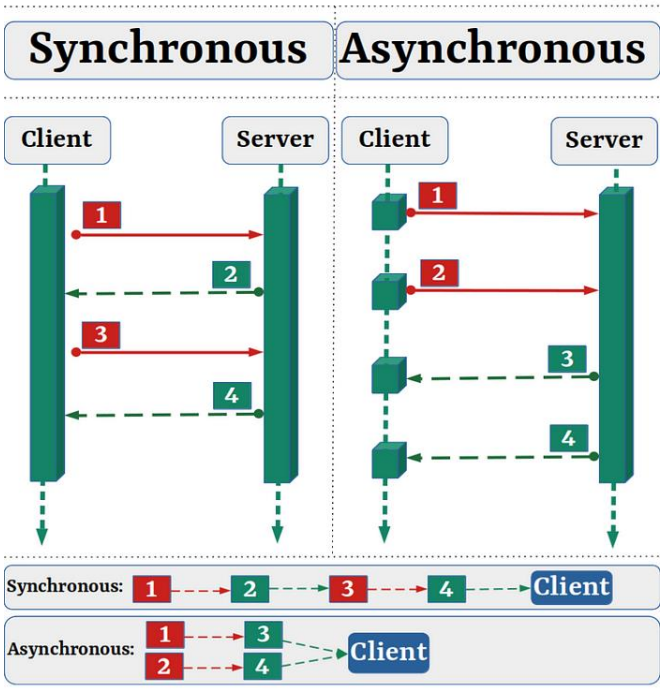


Рисунок 2.1 – Принципи синхронного та асинхронних відношень клієнт-сервер

Зокрема, для створення HTTP-сервера та здійснення асинхронних клієнтських запитів (наприклад, до API AbuseIPDB) була обрана бібліотека aiohttp. Вона надає всі необхідні інструменти для розробки високопродуктивного HTTP тарпіта, включаючи можливість потокової передачі даних (StreamResponse), що є ключовим для реалізації механізму уповільнення відповіді клієнту. Широка екосистема Python також надає готові рішення для роботи з базами даних (sqlite3), геолокацією (geoip2-python) та конфігурацією (python-dotenv).

Для зберігання зібраної інформації про зафіксовані події було вирішено використовувати локальну реляційну базу даних SQLite. Основними перевагами такого вибору є простота інтеграції (модуль sqlite3 є частиною стандартної бібліотеки

Python), відсутність необхідності розгортання окремого серверного процесу СКБД, та достатня продуктивність для очікуваних обсягів даних в рамках проекту. Вся база даних зберігається в одному файлі, що спрощує її резервне копіювання та перенесення. Хоча для дуже великих навантажень могли б розглядатися клієнт-серверні РСКБД (наприклад, PostgreSQL), для поточних завдань можливостей SQLite цілком достатньо [10]. Додатково, для цілей відладки та можливості інтеграції з зовнішніми системами аналізу логів, паралельно ведеться текстове логування у форматі JSON.

Збагачення даних про атакуючих здійснюється за допомогою двох ключових інструментів. Для визначення геолокації (країна, місто, координати) та інформації про автономну систему (ASN) використовуються локальні бази даних GeoLite2 від MaxMind у поєднанні з бібліотекою geoip2-python [11]. Такий підхід забезпечує швидкий пошук без необхідності онлайн-запитів для кожного IP-адресу. Для взаємодії з глобальною спільнотою з кібербезпеки та автоматичного звітування про шкідливі IP-адреси система інтегрується з сервісом AbuseIPDB через його API v2 [12]. Це дозволяє не тільки ділитися інформацією про виявлені загрози, але й потенційно отримувати дані про репутацію IP-адрес з цього сервісу.

Для управління залежностями проекту та забезпечення відтворюваності середовища розробки використовується інструмент Poetry [13]. Він дозволяє чітко визначати необхідні бібліотеки та їхні версії, ізолювати проектне середовище та спрощувати процес збірки та публікації пакета.

Останнім етапом буде аналіз отриманої інформації, та наочна візуалізація, за допомогою pandas та folium (рисунок 2.2).

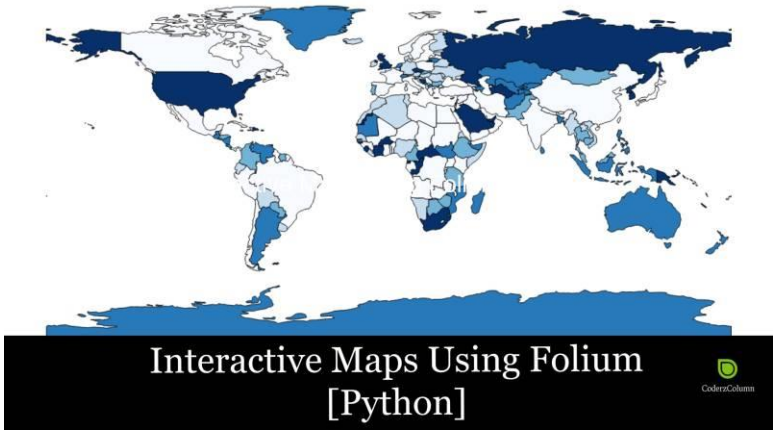


Рисунок 2.2 – Приклад інтерактивної візуалізації за допомогою folium

2.2 Розробка алгоритмів функціонування HTTP тарпіта та взаємодії з зовнішніми сервісами (GeoIP, AbuseIPDB)

Функціонування розробленої системи HTTP тарпіта визначається сукупністю взаємопов`язаних алгоритмів, що забезпечують обробку вхідних запитів, реалізацію механізму уповільнення, збагачення даних та взаємодію з зовнішніми сервісами для аналізу та звітування. Центральним елементом є алгоритм обробки HTTP-запиту, який можна деталізувати наступною послідовністю дій. Спочатку серверна частина тарпіта, реалізована на базі асинхронного фреймворку aiohttp, приймає вхідне TCP-з`єднання (рисунок 2.3) на попередньо налаштованому внутрішньому порту, куди надходять запити, проксійовані через Nginx.

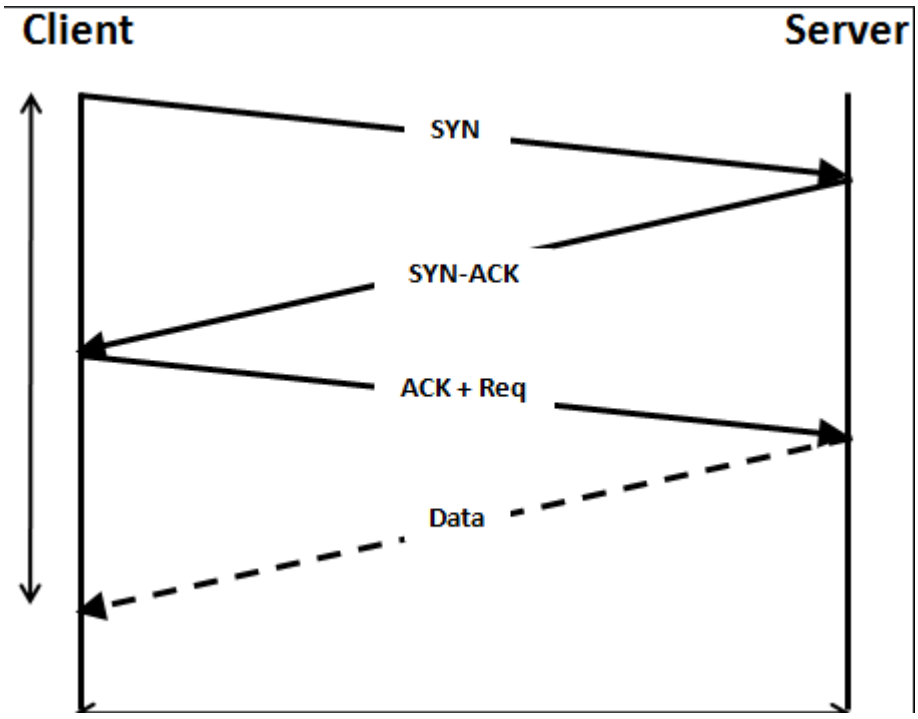


Рисунок 2.3 – Принцип трьохстороннього TCP handshake

Після встановлення з`єднання відбувається очікування та повний парсинг HTTP-запиту клієнта, під час якого виокремлюються ключові атрибути: HTTP-метод, запитуваний шлях, рядок параметрів запиту, версія протоколу та повний набір HTTP-заголовків, з особливою увагою до Host, User-Agent та Referer. Ця первинна інформація негайно фіксується.

Після ідентифікації IP-адреси клієнта відбувається етап збагачення даних. По-перше, здійснюється запит до локальних баз даних GeoLite2 City та GeoLite2 ASN від MaxMind за допомогою бібліотеки geoip2-python для отримання геолокаційної інформації (країна, місто, координати) та даних про автономну систему (номер ASN та організація-провайдер). Ця операція виконується синхронно, але в окремому потоці за допомогою asyncio.to_thread, щоб не блокувати основний асинхронний цикл. По-друге, ініціюється процес взаємодії з сервісом AbuseIPDB. Система перевіряє власну базу даних SQLite, чи не надходив звіт по даному IP-адресу протягом визначеного часового інтервалу (наприклад, остання година). Якщо IP не підпадає під критерії виключення (приватні мережі) і не був нещодавно зарепортований,

формується відповідний запит до API AbuseIPDB.

Наступний етап – безпосередня реалізація механізму тарпітингу. Система генерує стандартні HTTP-заголовки відповіді, наприклад, HTTP/1.1 200 OK та Content-Type: text/plain, які надсилаються клієнту максимально швидко через метод response.prepare(request) об`єкта aiohttp.StreamResponse. Це створює у атакуючого бота ілюзію успішної взаємодії з робочим сервером. Після цього починається уповільнена передача тіла відповіді: дані надсилаються невеликими порціями (наприклад, по одному байту) з конфігурованою затримкою між кожною порцією. Цей процес триває доти, доки не буде передано максимальний визначений обсяг даних або доки клієнт не розірве з`єднання. Протягом цього процесу система повинна коректно обробляти можливі помилки розриву з`єднання з боку клієнта, такі як ConnectionResetError або специфічне для aiohttp виключення ClientConnectionResetError, логуючи їх, але не припиняючи роботу обробника передчасно.

Після завершення передачі даних (або примусового розриву з`єднання) система намагається коректно завершити HTTP-відповідь викликом response.write_eof(). Уся зібрана під час сесії інформація, включаючи початкові дані запиту, реальний IP клієнта, дані GeoIP та ASN, інформацію про цільовий порт, тривалість сесії, кількість переданих байт, статус взаємодії (наприклад, чи був зв`язок розірваний клієнтом, чи всі дані були успішно відправлені) та статус спроби звітування в AbuseIPDB, записується у структурованому вигляді до бази даних SQLite та, для цілей відладки, до файлу логів у форматі JSON.

Алгоритм взаємодії з API сервісу AbuseIPDB також є важливою частиною системи. Після первинних перевірок (активність функції звітування, наявність API-ключа, відсутність IP у списку виключень та відсутність нещодавніх звітів по цьому IP з локальної БД) формується POST-запит до API AbuseIPDB. Запит містить IP-адресу, список категорій порушень (наприклад, сканування портів, атака на веб-додаток) та детальний коментар, що включає цільовий порт, запитуваний шлях,

HTTP-метод та User-Agent клієнта. Надсилання запиту реалізовано асинхронно за допомогою aiohttp.ClientSession та asyncio.create_task, щоб не блокувати основний потік роботи тарпіта. Система обробляє відповідь від AbuseIPDB, логуючи результат (успіх, помилка, зокрема 429 Too Many Requests) та, у випадку успішного звіту, фіксує спробу звіту та час у локальній базі даних для відповідної події.

2.3 Порівняльний аналіз архітектур розгортання HTTP тарпіта та інтеграції з існуючою інфраструктурою безпеки

Ефективність функціонування HTTP тарпіта значною мірою залежить не лише від його внутрішньої логіки, але й від обраної архітектури розгортання та способів інтеграції з існуючими компонентами інфраструктури безпеки. Розгляд різних сценаріїв розгортання дозволяє визначити оптимальний підхід, що забезпечує баланс між продуктивністю, безпекою та простотою адміністрування.

Одним з базових варіантів є розгортання HTTP тарпіта як повністю автономного сервісу, що безпосередньо прослуховує публічні порти 80 (HTTP) та 443 (HTTPS) на виділеному IP-адресі. Такий підхід може бути виправданий для створення ізольованої "приманки" або в дуже обмежених середовищах. Однак, він має суттєві недоліки з точки зору безпеки та управління. По-перше, для прослуховування привілейованих портів (нижче 1024) Python-додаток потребуватиме запуску з правами суперкористувача (root) або надання йому спеціальних можливостей ядра (наприклад, CAP_NET_BIND_SERVICE). Запуск всього додатку з підвищеними привілеями створює значні ризики безпеки: будь-яка потенційна вразливість у коді тарпіта може призвести до компрометації всієї системи. По-друге, реалізація SSL/TLS-термінації безпосередньо в Python-додатку за допомогою aiohttp хоча й можлива, але є менш ефективною та більш складною в управлінні сертифікатами порівняно зі спеціалізованими вебсерверами.

Значно більш поширеною та рекомендованою архітектурою є розгортання

HTTP тарпіта за зворотним проксі-сервером (reverse proxy), таким як Nginx або Apache. У цьому сценарії зворотний проксі-сервер приймає весь вхідний трафік на публічних портах 80 та 443, виконує SSL/TLS-термінацію (обробку HTTPS), а потім перенаправляє HTTP-запити на внутрішній порт (наприклад, 8080), де їх прослуховує Python-додаток тарпіта, запущений від імені звичайного, непривілейованого користувача (рисунок 2.4).

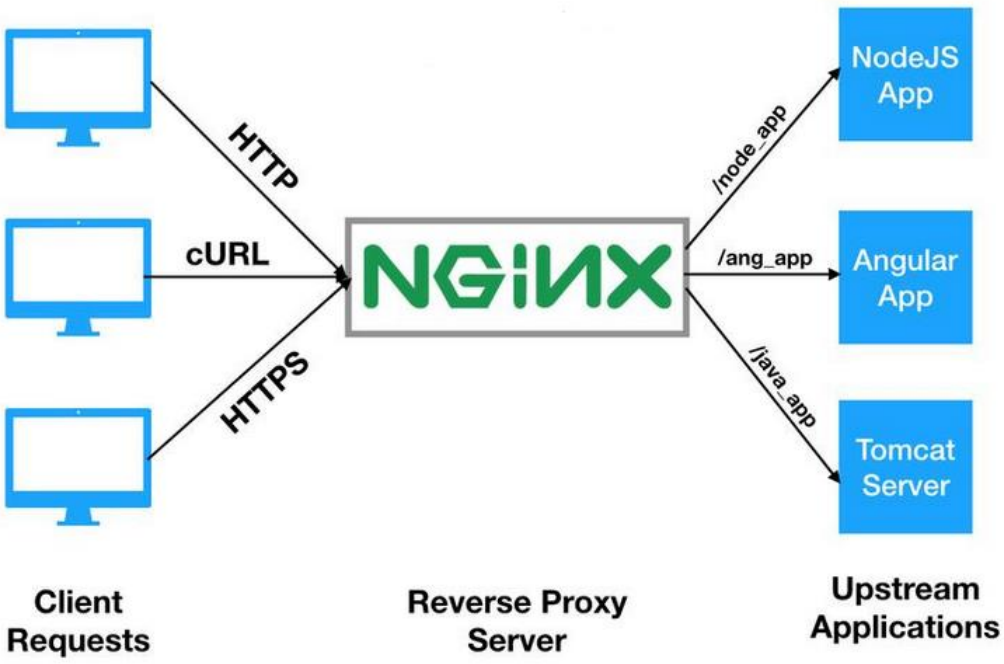


Рисунок 2.4 – Зворотній проксі-сервер на Nginx

Така архітектура надає низку ключових переваг. З точки зору безпеки, основний Python-додаток працює з мінімально необхідними правами, що істотно знижує потенційну поверхню атаки. Крім того, Nginx, як спеціалізований вебсервер, має вбудовані механізми захисту від деяких типів атак (наприклад, повільних запитів типу Slowloris, хоча це менш актуально для тарпіта, який сам уповільнює), а також дозволяє легко налаштовувати базові правила фільтрації. Ефективність обробки SSL/TLS значно вища, а управління сертифікатами (наприклад, за допомогою Certbot для Let`s

Encrypt) спрощується завдяки інтеграції certbot з Nginx. Nginx також ефективно управляє великою кількістю одночасних з`єднань та може виконувати кешування або балансування навантаження (якщо тарпіт масштабується на кілька екземплярів). Для коректної роботи тарпіта за Nginx важливо правильно налаштувати передачу реальної IP-адреси клієнта (через заголовки X-Forwarded-For та X-Real-IP) та інформації про протокол (X-Forwarded-Proto). Також критичним для збереження ефекту "заморожування" є відключення буферизації відповіді на стороні Nginx (proxy_buffering off;), щоб дані від тарпіта негайно передавалися клієнту.

Важливим аспектом безпеки при розгортанні будь-якого веб-додатку, включаючи тарпіт, є управління конфігураційними даними та секретами, такими як API-ключі (наприклад, для AbuseIPDB). Зберігання таких даних безпосередньо в коді програми або в системі контролю версій є неприпустимою практикою. Рекомендованим підходом є використання файлів змінних оточення (наприклад, .env файлу) у поєднанні з бібліотекою python-dotenv. Файл .env розміщується на сервері в кореневій директорії проекту, містить пари ключ-значення (наприклад, ABUSEIPDB_API_KEY=your_secret_key) і додається до файлу .gitignore, щоб уникнути його потрапляння до репозиторію. Додаток при старті завантажує ці змінні в своє оточення, звідки їх можна безпечно читати. Це забезпечує розділення коду та конфігурації, а також захист чутливої інформації.

Інтеграція HTTP тарпіта з існуючою інфраструктурою безпеки також є важливим аспектом. Зібрані тарпітом дані (IP-адреси, User-Agent, шляхи) можуть передаватися до SIEM-систем (Security Information and Event Management) для кореляції з іншими подіями безпеки та більш глибокого аналізу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ HTTP ТАРПІТА

3.1 Обґрунтування вибору інструментальних засобів та технологій розробки

Процес розробки програмного засобу HTTP тарпіта розпочався з налаштування робочого середовища та управління залежностями проекту, для чого було обрано інструмент Poetry. Ініціалізація проекту за допомогою Poetry (poetry new http-tarpit) створює стандартизовану структуру директорій, включаючи основну папку для коду src/http_tarpit/ та файл pyproject.toml. Цей файл є центральним для конфігурації проекту, визначення його метаданих (назва, версія, автори), залежностей Python та скриптів командного рядка. Усі залежності, такі як aiohttp для асинхронного вебсервера, geoip2-python для геолокації, python-dotenv для управління конфігурацією та pandas для майбутнього аналізу даних, були додані до проекту за допомогою команди poetry add <package_name>, що забезпечило їх фіксацію у файлі poetry.lock та створення ізольованого віртуального оточення. Такий підхід гарантує відтворюваність середовища розробки та розгортання. Отож маємо таку структуру:

```
http-tarpit/
├── src/
│   ├──http-tarpit/
├── main.py
├── README.md
├── poetry.lock
└── pyproject.toml
```

Після налаштування проекту розпочалася реалізація ядра HTTP тарпіта, основною метою якого є прийом HTTP-запитів та їх навмисно уповільнена обробка. Серверна частина була реалізована з використанням асинхронного фреймворку aiohttp на базі стандартного модуля asyncio Python. Точкою входу для запуску сервера є файл main.py, який ініціалізує необхідні компоненти, такі як система логування та база даних, а потім викликає основну функцію запуску сервера, розташовану в модулі src/http_tarpit/tarpit_server.py.

У модулі tarpit_server.py функція run_server() відповідає за створення та

конфігурацію екземпляра aiohttp.web.Application. До цього екземпляра додається єдиний маршрут, що використовує шаблон /{tail:.*} та метод * (будь-який HTTP-метод), який скеровує всі вхідні запити на центральну функцію-обробник handle_request з модуля request_handler.py. Фрагмент коду, що демонструє створення додатку та додавання маршруту:

```
async def run_server():
    app = web.Application()
    app.router.add_route(`*`, `/{tail:.*}`, handle_request)
```

Для фактичного запуску сервера та прослуховування мережевого порту використовуються об`єкти aiohttp.web.AppRunner та aiohttp.web.TCPSite. TCPSite конфігурується для прослуховування хоста та порту, визначених у модулі config.py (наприклад, 127.0.0.1:8080 для роботи за зворотним проксі). При старті сервер логує інформацію про свою готовність та основні параметри тарпітингу. Для забезпечення безперервної роботи сервер запускається в асинхронному циклі, який підтримує його активність. Нижче наведено фрагмент коду, що відповідає за запуск та підтримку роботи сервера, увесь код tarpit_server.py та request_handler.py буде надано у Додатку А:

```
    runner = web.AppRunner(app)
        await runner.setup()
        site = web.TCPSite(runner, config.HOST, config.PORT)
        log.info(f"Attempting to start HTTP Tarpit server on
http://{config.HOST}:{config.PORT}")
        log.info(f"Tarpit                                settings:
Delay={config.RESPONSE_DELAY_SECONDS}s,
Chunk={config.RESPONSE_CHUNK!r},
MaxBytes={config.MAX_RESPONSE_BYTES}")
        try:
            await site.start()
            log.info("Server  started  successfully.  Waiting
for connections...")
            while True:
                await asyncio.sleep(3600)
        except Exception as e:
            log.exception("Failed to start or run the server")
        finally:
```

Особливу увагу приділено коректному завершенню роботи сервера у блоці finally, де відбувається зупинка TCPSite та очищення ресурсів AppRunner, що важливо для стабільної роботи в довгостроковій перспективі та при перезапусках сервісу.

Центральна логіка обробки кожного HTTP-запиту та реалізація механізму уповільнення ("тарпітингу") зосереджена в асинхронній функції handle_request модуля src/http_tarpit/request_handler.py. При отриманні нового запиту ця функція спочатку збирає всю необхідну інформацію про клієнта (визначаючи реальну IP-адресу з урахуванням заголовків X-Forwarded-For/X-Real-IP, що додаються Nginx) та сам запит (метод, шлях, заголовки, User-Agent).

Ключовим для реалізації ефекту "заморожування" є використання об`єкта aiohttp.web.StreamResponse. На відміну від стандартного aiohttp.web.Response, StreamResponse дозволяє надсилати HTTP-відповідь потоково. Спочатку, за допомогою await response.prepare(request), клієнту негайно надсилаються HTTP-заголовки (наприклад, 200 OK, Content-Type: text/plain, Connection: keep-alive). Це створює у клієнта (бота) ілюзію швидкої відповіді від сервера. Після цього починається ітераційна передача тіла відповіді. В асинхронному циклі while генерується невелика порція даних (наприклад, один байт b`.`), яка надсилається клієнту через await response.write(). Для гарантії фактичної передачі даних перед паузою використовується await response.drain(). Між відправками кожної порції даних вставляється асинхронна пауза await asyncio.sleep(config.RESPONSE_DELAY_SECONDS). Цей процес триває до передачі максимальної визначеної кількості байт (config.MAX_RESPONSE_BYTES) або до розриву з`єднання клієнтом. Обробка винятків, таких як ConnectionResetError та ClientConnectionResetError, інтегрована для логування цих очікуваних подій без переривання роботи основного обробника. Фіналізація відповіді відбувається викликом await response.write_eof().

3.2 Реалізація модулів збору, збагачення, зберігання даних та взаємодії з зовнішніми сервісами

Окрім ядра HTTP тарпіта, що відповідає за прийом та уповільнену обробку запитів, розроблена система включає низку важливих модулів, які забезпечують збір, збагачення, зберігання даних про атаки, а також взаємодію з зовнішніми сервісами для підвищення інформативності та ефективності протидії. Загалом, з реалізованими усіма модулями структура проекту буде виглядати так:

```
http-tarpit/
├── src/
│   ├──http-tarpit/
│   ├──reportint/
│   │   ├──abuseipdb_reporter.py
│   ├──utils/
│   │   ├──__init.py
│   │   ├──geoip_lookup.py
│   ├──__init.py
│   ├──config.py
│   ├──database.py
│   ├──logger_setup.py
│   ├──request_handler.py
│   ├──tarpit_server.py
├── main.py
├── README.md
├── poetry.lock
└── pyproject.toml
```

Модуль конфігурації (config.py) слугує централізованим місцем для зберігання всіх налаштувань програми. Це включає мережеві параметри (хост та порт для внутрішнього сервера aiohttp), параметри механізму уповільнення (RESPONSE_DELAY_SECONDS, RESPONSE_CHUNK, MAX_RESPONSE_BYTES), шляхи до файлів логів, бази даних SQLite та баз GeoIP, а також налаштування для інтеграції з AbuseIPDB (API-ключ, пороговий рівень впевненості, категорії звітів). Для безпечного зберігання API-ключа використовується бібліотека python-dotenv, яка завантажує змінні оточення з файлу .env, що не включається до системи контролю

версій. Такий підхід забезпечує гнучкість налаштування та відокремлення конфігурації від кодової бази. При ініціалізації модуля конфігурації також перевіряється наявність необхідних директорій (для логів, даних) та, за потреби, вони створюються.

Система логування (logger_setup.py) побудована на базі стандартного модуля logging Python. Для забезпечення структурованості логів та їхньої придатності для подальшого автоматизованого аналізу було розроблено кастомний форматер JsonFormatter. Цей форматер перетворює кожне лог-повідомлення на JSON-об`єкт, що включає стандартні поля (час, рівень, ім`я логгера, повідомлення), а також будь-які додаткові дані, передані через параметр extra при виклику логуючої функції. Налаштовуються два обробники: FileHandler для запису JSON-логів у файл, шлях до якого визначається в конфігурації, та StreamHandler для виведення текстових лог-повідомлень (зазвичай рівня INFO та вище) у стандартний потік виводу (консоль), що зручно для моніторингу роботи сервісу systemd через journalctl. Рівні логування для файлу та консолі також конфігуруються окремо.

Модуль роботи з базою даних (database.py) інкапсулює всю логіку взаємодії з локальною базою даних SQLite. При першому запуску програми викликається функція init_db(), яка створює файл бази даних (шлях визначається в config.py) та таблицю events зі структурою, детально описаною в підрозділі 2.3, якщо вони ще не існують. Реалізовано механізм безпечного додавання нових колонок до існуючої таблиці за допомогою ALTER TABLE для забезпечення сумісності при оновленні схеми. Основна функція log_event_to_db(event_data: dict) приймає словник з даними про подію та записує їх у таблицю events за допомогою параметризованого SQL-запиту INSERT, що запобігає SQL-ін`єкціям. Оскільки операції з sqlite3 є синхронними, виклик log_event_to_db з основного асинхронного коду обробника запитів здійснюється через await asyncio.to_thread(), що дозволяє виконати блокуючу операцію в окремому потоці, не зупиняючи цикл подій asyncio. Також у цьому модулі реалізовані функції check_ip_reported_recently(ip_address: str, interval_hours: int) для

перевірки, чи був IP-адрес зарепортований до AbuseIPDB протягом заданого інтервалу (за даними з локальної БД), та mark_ip_as_reported(ip_address: str, event_id: int) для оновлення статусу репорту для конкретної події

Модуль геолокації (utils/geoip_lookup.py) відповідає за визначення географічного розташування та інформації про автономну систему (ASN) за IP-адресою клієнта. При ініціалізації модуля завантажуються локальні бази даних GeoLite2 City та GeoLite2 ASN від MaxMind (шляхи до файлів беруться з config.py) за допомогою бібліотеки geoip2-python. Функція get_geoip_data(ip_address: str) приймає IP-адресу, здійснює пошук у завантажених базах та повертає словник з даними (країна, місто, координати, номер ASN, організація ASN) або порожній словник, якщо IP-адреса приватна, локальна або не знайдена в базах. Обробка помилок, таких як AddressNotFoundError, включена для забезпечення стабільності. Ця функція також викликається через await asyncio.to_thread() з обробника запитів.

Модуль звітування в AbuseIPDB (reporting/abuseipdb_reporter.py) реалізує асинхронну функцію report_ip_to_abuseipdb(ip_address: str, target_port: int, comment_details: str). Вона формує POST-запит до API v2 сервісу AbuseIPDB, що містить IP-адресу, список категорій порушень (визначених у config.py) та коментар, який включає цільовий порт, запитуваний шлях, HTTP-метод та User-Agent. Запит надсилається за допомогою aiohttp.ClientSession, що забезпечує неблокуючу мережеву взаємодію. Функція обробляє відповіді від API, логуючи успішні репорти (включаючи новий abuseConfidenceScore) та помилки (наприклад, перевищення лімітів 429 Too Many Requests, невалідний API-ключ 401 Unauthorized).

Програмний код цих файлів буде надано в Додатку А та в моєму GitHub [14], але загалом, побудувавши внутрішню архітектуру її можна візуалізувати наступним чином (рисунок 3.1).

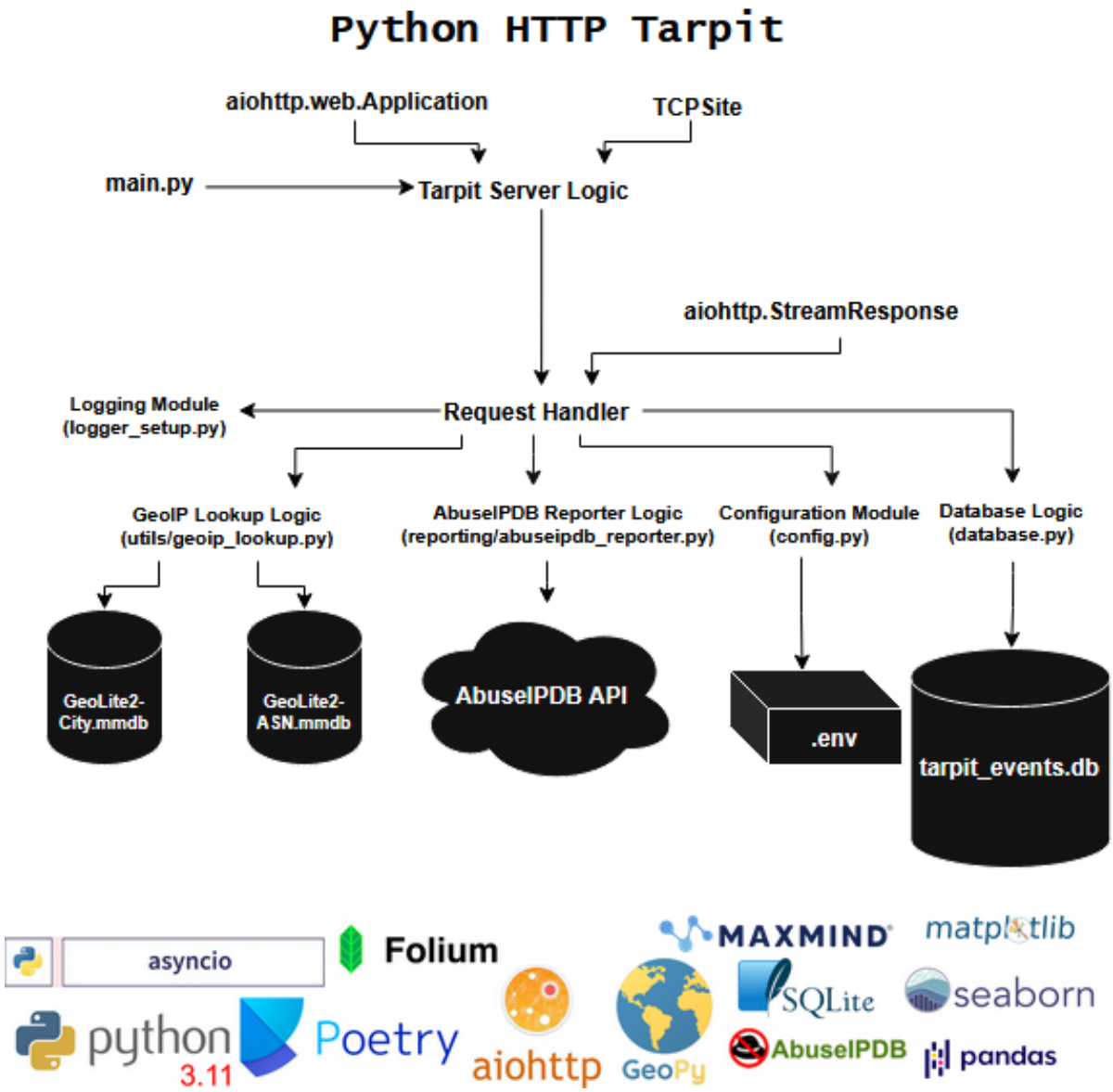


Рисунок 3.1 – Внутрішня архітектура та технологочній стек HTTP Tarpit

Архітектура розгорнутої на VPS системи, буде виглядати наступним чином (рисунку 3.2).

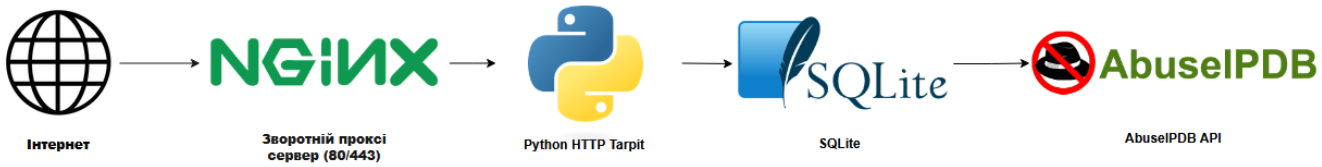


Рисунок 3.2 – Архітектура розгорнутої на VPS системи (HTTP Tarpit)

3.3 Тестування та розгортання програмного засобу

Після завершення програмної реалізації всіх компонентів HTTP тарпіта було проведено комплексне тестування з метою перевірки його функціональності, стабільності роботи та відповідності поставленим вимогам. Тестування охоплювало як локальну перевірку окремих модулів та загальної логіки, так і розгортання системи на віртуальному приватному сервері (VPS) для оцінки її поведінки в умовах, наближених до реальних інтернет-загроз.

Локальне тестування проводилося на віртуальній машині. На цьому етапі перевірялися:

Коректність роботи механізму уповільнення: За допомогою HTTP-клієнтів, таких як curl (рисунок 3.2), здійснювалися запити до локально запущеного тарпіта. Фіксувався час відповіді, побайтова передача даних та загальна тривалість утримання з`єднання відповідно до конфігураційних параметрів (RESPONSE_DELAY_SECONDS, MAX_RESPONSE_BYTES).

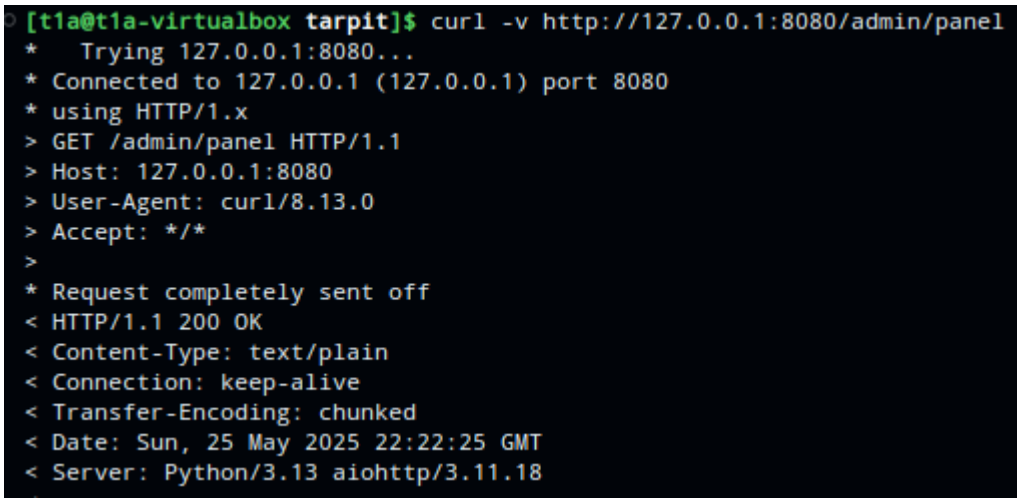


Рисунок 3.2 – Запит за допомогою curl до локального HTTP тарпіту

Правильність збору даних: Аналізувався вміст JSON-логів (рисунок 3.3) та бази даних SQLite (рисунок 3.4) після кожної тестової сесії. Перевірялася повнота та

коректність записуваних даних: IP-адреси, HTTP-заголовків, User-Agent, шляхів, а також даних, отриманих від модулів GeoIP та AbuseIPDB (для тестових IP, що не належать до приватних мереж).

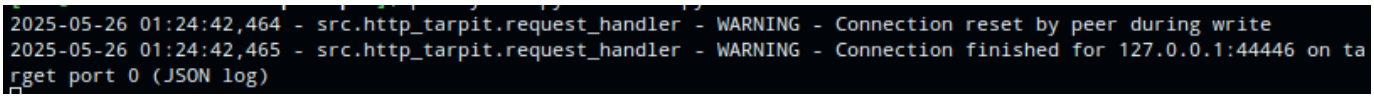


Рисунок 3.3 – Запис в логах після минулого curl запиту

client_ip	client_port	http_method	http_path	user_agent	headers_json	response_status	bytes_sent	duration_s
Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
93.123.109.229	54224	GET	/.env.save	l9explore/1.2.2	{"Host": "46.101.106.95", "X-Real-…	200	4	6.026

Рисунок 3.4 – Скорочений запис в базу даних SQLite після минулого curl запиту

Функціонування інтеграції з AbuseIPDB: Здійснювалися тестові запити з IP-адрес, що не підпадають під виключення, для перевірки логіки формування та асинхронного надсилання звітів. Аналізувалися логи модуля abuseipdb_reporter.py на предмет успішної взаємодії з API (рисунок 3.5).

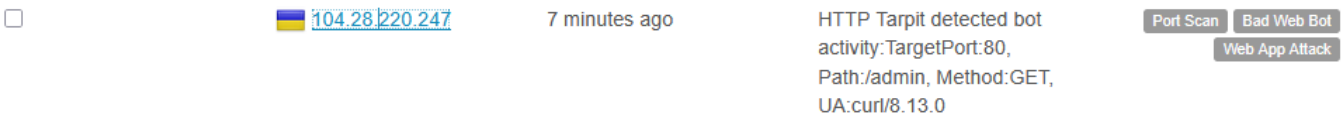


Рисунок 3.5 – Сформований репорт за допомогою AbuseIPDB API після минулого curl запиту

Розгортання на віртуальному приватному сервері (VPS) було здійснено на платформі DigitalOcean (рисунок 3.6) з використанням операційної системи Debian. Процес розгортання включав клонування проекту з Git-репозиторію, встановлення залежностей за допомогою Poetry, налаштування файлу .env з API-ключем AbuseIPDB, розміщення локальних баз GeoIP.

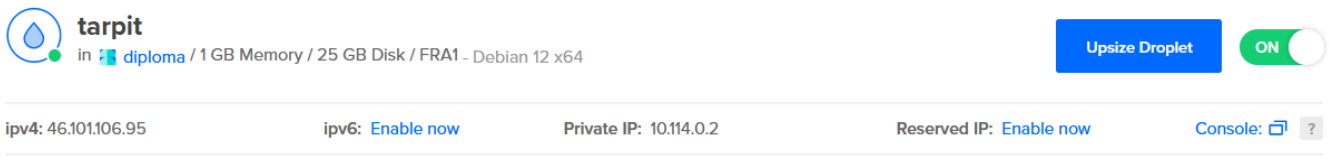


Рисунок 3.6 – VPS на якому цілодобово хоститься HTTP тарпіт

Nginx було налаштовано як зворотний проксі, що приймає вхідний трафік на портах 80 (HTTP) та 443 (HTTPS), та додані правила для фаєрволу UFW (рисунок 3.7).

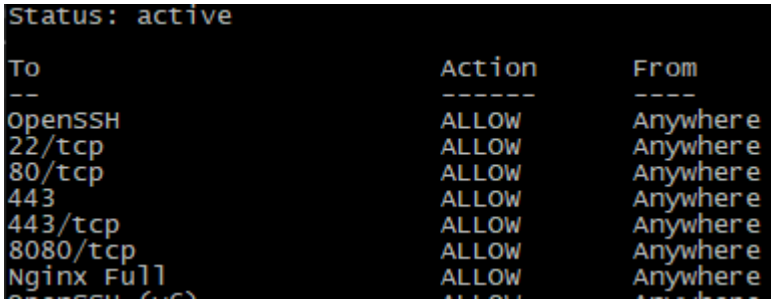


Рисунок 3.7 – Сформований репорт за допомогою AbuseIPDB API після минулого curl запиту

Було придбано доменне ім`я (pastka.xyz) на агригаторі NameSilo (рисунок 3.8), на домені були налаштовані DNS під ip-адреси VPS. Тепер nginx виконує SSL/TLS-термінацію (з використанням сертифікатів Let`s Encrypt, отриманих через Certbot) та перенаправляє HTTP на HTTPS.

Рисунок 3.8 – Зареєстроване доменне ім`я для HTTPS

Запуск Python-додатку було автоматизовано за допомогою systemd (рисунок 3.9) сервісу, що забезпечує його роботу у фоновому режимі, автоматичний перезапуск у випадку збоїв та управління через стандартні команди systemctl.

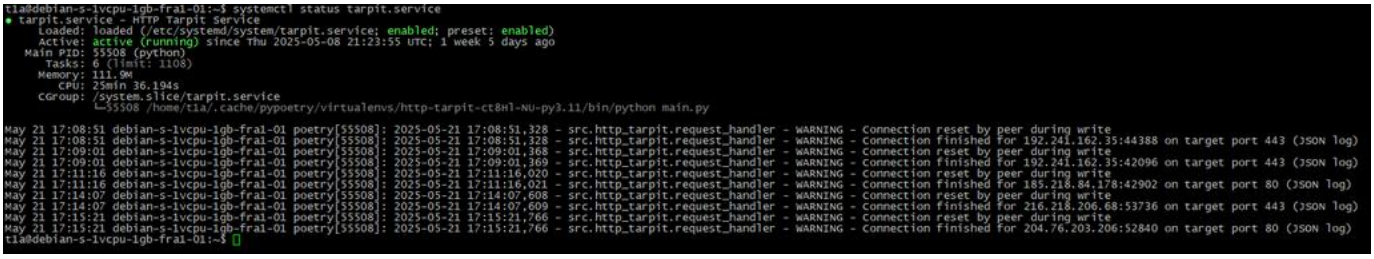


Рисунок 3.9 – Робота HTTP тарпіту за допомогою systemd сервісу

Після розгортання на VPS з використанням Nginx та запуску як systemd-сервісу, HTTP тарпіт було залишено для збору даних про реальні інтернет-загрози. За період з 8 травня 2025 року по 24 травня 2025 року було зафіксовано 27 340 подій (рисунок 3.10) від 5066 унікальних IP-адрес (рисунок 3.11).

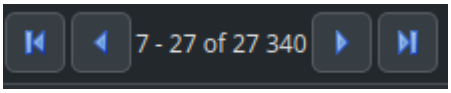


Рисунок 3.10 – Кількість записів у базу даних

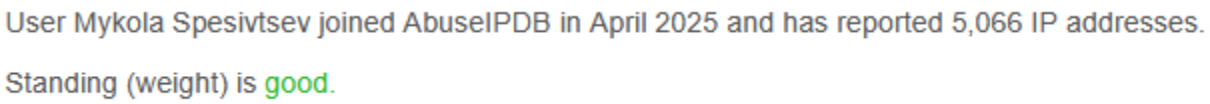


Рисунок 3.11 – Кількість зарепорчених ip-адресів на AbuseIPDB

Також за допомогою pandas, matplotlib, та folium ми можемо побачити деякі візуалізації зібраної інформації, які будуть надані в Додатку Б.

ВИСНОВОК

У ході виконання даної дипломної роботи було успішно вирішено поставлене завдання щодо розробки та дослідження програмного засобу типу "ресурсомістка пастка" (HTTP Tarpit), призначеного для ускладнення автоматизованих атак на вебсервери, збору інформації про атакуючих та інтеграції з сервісами репутації IP-адрес.

На першому етапі було проведено детальний аналіз предметної області, включаючи вивчення сучасних автоматизованих загроз для вебсерверів, таких як сканування вразливостей, брутфорс-атаки, DDoS-атаки рівня додатку та активність шкідливих ботів. Було розглянуто існуючі методи та засоби протидії, виявлено їхні переваги та недоліки, що обґрунтувало актуальність розробки активних методів захисту, зокрема HTTP тарпітів.

На етапі теоретичного проектування було сформульовано функціональні та нефункціональні вимоги до розроблюваної системи. Було обрано архітектуру, що передбачає роботу Python-додатку за зворотним проксі-сервером Nginx, та обґрунтовано вибір технологічного стеку, ключовими елементами якого стали Python, асинхронний фреймворк aiohttp, база даних SQLite, бібліотека geoip2-python для роботи з базами GeoLite2 та API сервісу AbuseIPDB. Було розроблено алгоритми функціонування основного механізму уповільнення, збору та збагачення даних, а також взаємодії із зовнішніми сервісами. Також було спроектовано структуру бази даних для зберігання інформації про зафіксовані події.

На етапі програмної реалізації було створено багатомодульний Python-додаток, що включає ядро HTTP тарпіта з механізмом уповільненої потокової відповіді, модулі для роботи з конфігурацією, логуванням у форматі JSON, взаємодії з базою даних SQLite, геолокації IP-адрес та автоматичного звітування в AbuseIPDB. Особливу увагу було приділено асинхронній обробці запитів для забезпечення високої продуктивності та здатності обробляти значну кількість одночасних повільних з`єднань.

Розроблений програмний засіб було успішно розгорнуто на віртуальному приватному сервері (VPS) під управлінням ОС Debian. Було налаштовано вебсервер Nginx як зворотний проксі для обробки трафіку на портах 80 (HTTP) та 443 (HTTPS), реалізовано SSL/TLS-термінацію за допомогою сертифікатів Let`s Encrypt та автоматичне перенаправлення HTTP на HTTPS. Python-додаток тарпіта було запущено як системний сервіс за допомогою systemd, що забезпечує його стабільну роботу та автоматичний перезапуск. Налаштовано фаєрвол UFW для контролю доступу.

У ході тестової експлуатації на VPS система продемонструвала свою працездатність: тарпіт успішно приймав та уповільнював запити від реальних інтернет-ботів та сканерів, збирав детальну інформацію про них, включаючи геолокаційні дані та дані про ASN, та здійснював спроби звітування до AbuseIPDB. Аналіз зібраних даних (на прикладі окремих записів) підтвердив здатність системи ідентифікувати типові патерни автоматизованої шкідливої активності.

Таким чином, мета дипломної роботи досягнута: розроблено та апробовано програмний засіб HTTP тарпіт, який є функціональним інструментом для активної протидії автоматизованим загрозам. Він дозволяє не лише ускладнити діяльність ботів шляхом вичерпання їхніх ресурсів, але й збирати цінну розвідувальну інформацію, яка може бути використана для подальшого аналізу, покращення систем безпеки та проактивного блокування шкідливих джерел.

Подальший розвиток проекту може включати реалізацію модуля поглибленого аналізу та візуалізації зібраних даних, вдосконалення механізмів уповільнення, розширення типів "приманок" для ботів та інтеграцію з іншими системами безпеки, такими як SIEM або WAF.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бойко В.Д., Спесівцев М.А. Використання ресурсомістких пасток для ускладнення атаки на вебсервер // Кібербезпека в сучасному світі: актуальні виклики. Матеріали V Міжнародної науково-практичної конференції (м. Одеса 22 листопада 2024р.). 2024. P. 28-31.
2. BasuMallick C. Network Security What Is Botnet? Definition, Methods, Attack Examples, and Prevention Best Practices for 2022. Network Security. URL: https://www.spiceworks.com/it-security/network-security/articles/what-is-botnet/.
3. Honcharenko D. 10 вразливостей WordPress, про які варто знати. Захист сайту на WordPress. WordPress. URL: https://hostpro.ua/blog/ua/ten-wordpress-vulnerabilities/.
4. Walter D. Why Modern Layer 7 DDoS Protections Are Crucial for Web Security in 2024. Security. URL: https://www.akamai.com/blog/security/why-modern-layer-7-ddos-protections-crucial-web-security-2024.
5. Бойко В.Д., Василенко М.Д., Слатвінська В.М. Версіонування файлової системи для боротьби з програмами-вимагачами // Інформатика, управління та штучний інтелект Тези восьмої міжнародної науково-технічної конференції (16 – 19 листопада 2021 року). Харків: НТУ "ХПІ", 2021. P. 9.
6. Бойко В., Василенко М., Слатвінська В. Моделювання живучості та відновлення інформаційно-комунікаційних мереж в умовах дії кіберзагроз // Інформаційні технології та суспільство. 2024. № 1 (12). P. 13–19.
7. Ngila F. AI bots are so good at mimicking the human brain and vision that CAPTCHAs are useless. A.I. URL: https://qz.com/ai-bots-recaptcha-turing-test-websites-authenticity-1850734350.
8. Debatty T. Using LaBrea Tarpit to hinder network scans. Blog. URL: https://cylab.be/blog/29/using-labrea-tarpit-to-hinder-network-scans.
9. The Tarpit: A Cybersecurity Trap for the Unwary. HeyCoach. URL: https://blog.heycoach.in/tarpit/.

10. SQLite Documentation. SQLite Home Page. URL: https://www.sqlite.org/docs.html.
11. GeoLite Databases and Web Services. MaxMind. URL: https://dev.maxmind.com/geoip/geolite2-free-geolocation-data/.
12. Introduction to API V2. AbuseIPDB. URL: https://docs.abuseipdb.com/#introduction.
13. Poetry - Python dependency management and packaging made easy. Poetry. URL: https://python-poetry.org/docs/.
14. Spesivtsev Mykola. HTTP Tarpit & Bot Analyzer. Version 1.0. URL: https://github.com/t1a0/http-tarpit.

Додаток А. Програмний код

Файл tarpit_server.py

```
import asyncio
import logging
from aiohttp import web

from .request_handler import handle_request
from . import config

log = logging.getLogger(__name__)

async def run_server():
    app = web.Application()
    app.router.add_route('*', '/{tail:.*}', handle_request)

    runner = web.AppRunner(app)
    await runner.setup()
    site = web.TCPSite(runner, config.HOST, config.PORT)

    log.info(f"Attempting to start HTTP Tarpit server on
http://{config.HOST}:{config.PORT}")
    log.info(f"Tarpit settings:
Delay={config.RESPONSE_DELAY_SECONDS}s,
Chunk={config.RESPONSE_CHUNK!r}, MaxBytes={config.MAX_RESPONSE_BYTES}")

    try:
        await site.start()
        log.info("Server started successfully. Waiting for
connections...")
        while True:
            await asyncio.sleep(3600)
    except Exception as e:
        log.exception("Failed to start or run the server")
        await runner.cleanup()
        raise
    finally:
        log.info("Shutting down server resources...")
        if 'site' in locals() and site._server is not None:
            log.info("Stopping TCPSite...")
            await site.stop()
            log.info("TCPSite stopped.")
        if 'runner' in locals():
            log.info("Cleaning up AppRunner...")
            await runner.cleanup()
            log.info("AppRunner cleaned up.")
        log.info("Server resources shut down.")
```

Файл config.py

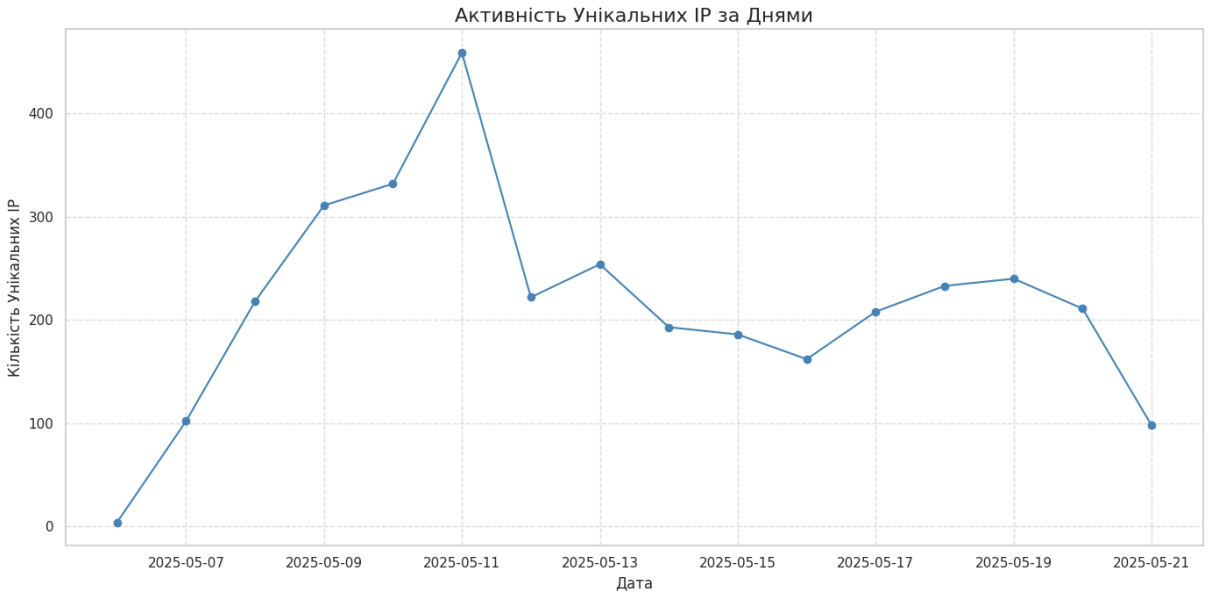
```
import logging
import os
from pathlib import Path
from dotenv import load_dotenv

BASE_DIR = Path(__file__).resolve().parent.parent.parent

load_dotenv(dotenv_path=BASE_DIR/'.env')

HOST = "127.0.0.1"
PORT = 8080
LOG_DIR = BASE_DIR / "logs"
LOG_FILE = LOG_DIR / "tarpit.log"
LOG_LEVEL = logging.INFO
CONSOLE_LOG_LEVEL = logging.WARNING

# bd config
DATABASE_DIR = BASE_DIR / "data"
SQLITE_DB_FILE = DATABASE_DIR / "tarpit_events.db"

# tarpit config
RESPONSE_DELAY_SECONDS = 1.5
RESPONSE_CHUNK = b'.'
MAX_RESPONSE_BYTES = 1200

ABUSEIPDB_API_KEY = os.getenv("ABUSEIPDB_API_KEY",None)
ABUSEIPDB_ENABLED = bool(ABUSEIPDB_API_KEY)
ABUSEIPDB_CONFIDENCE_SCORE = 90
ABUSEIPDB_COMMENT_PREFIX = "HTTP Tarpit detected bot activity:"
ABUSEIPDB_CATEGORIES = "14,21,19" # 14 = Port Scan, 21 = Web App
Atack, 19 = Bad Web Bot (?18 = Brute?)
ABUSEIPDB_REPORT_INTERVAL_MINUTES = 40

GEOLITE2_CITY_DB_PATH = BASE_DIR / "data" / "GeoLite2-City.mmdb"
GEOLITE2_ASN_DB_PATH = BASE_DIR / "data" / "GeoLite2-ASN.mmdb"

GEOIP_CITY_ENABLED = GEOLITE2_CITY_DB_PATH.exists()
GEOIP_ASN_ENABLED = GEOLITE2_ASN_DB_PATH.exists()

LOG_DIR.mkdir(parents=True, exist_ok=True)
DATABASE_DIR.mkdir(parents=True, exist_ok=True)
```

Файл abuseipdb_reporter.py

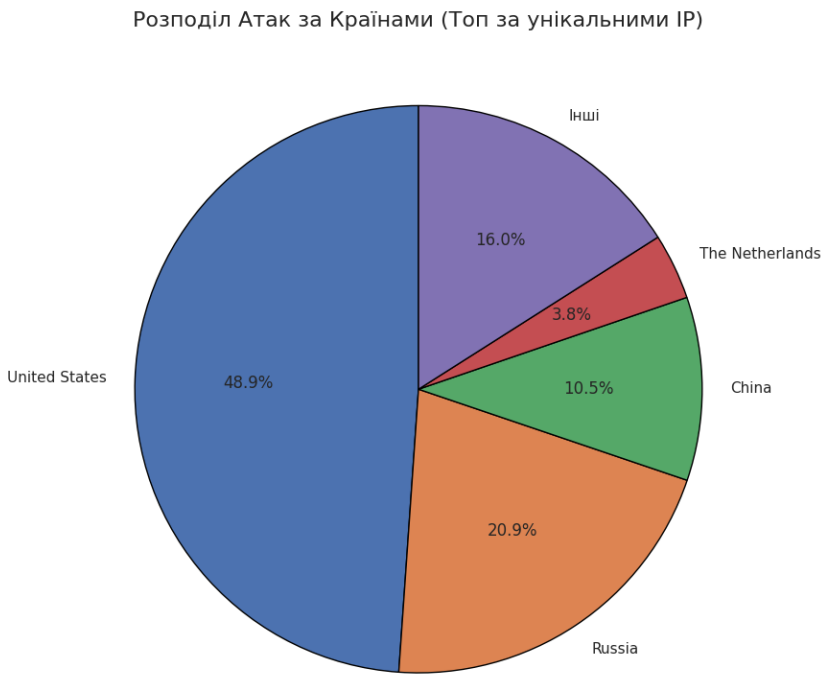
```
    import logging
    import asyncio
    from aiohttp import ClientSession, ClientError,
ClientResponseError

    from .. import config

    log = logging.getLogger(__name__)

    ABUSEIPDB_API_URL = 'https://api.abuseipdb.com/api/v2/report'


    async def report_ip_to_abuseipdb(ip_address: str,  target_port:
int, comment_details: str):
        """Асинхронная отправка репорта в AbuseIPDB"""
        if not config.ABUSEIPDB_ENABLED:
            log.debug("AbuseIPDB reporting is disabled in config.")
            return
        if not config.ABUSEIPDB_API_KEY:
            log.error("AbusIPDB reporting is enablud, but API is
missing!")
            return

        headers = {
            'Accept': 'application/json',
            'Key': config.ABUSEIPDB_API_KEY
        }

        params = {
            'ip': ip_address,
            'categories': config.ABUSEIPDB_CATEGORIES,
            'comment':
f"{config.ABUSEIPDB_COMMENT_PREFIX}{comment_details}"
        }

        log.info(f"Attempting to report IP {ip_address} (from target
port {target_port}) to AbuseIPDB. Categories: {params['categories']}")

        try:
            async with ClientSession(headers=headers) as session:
                async with session.post(ABUSEIPDB_API_URL,
data=params) as response:
                    response_json = await response.json()

                    response.raise_for_status()
```

```
                    response_json = await response.json()

                    abuse_data = response_json.get('data', {})
                    score = abuse_data.get('abuseConfidenceScore')

                    if score is not None:
                        log.info(f"Successfully reported IP
{ip_address} (from target port {target_port}) to AbuseIPDB. New
confidence score: {score}")
                    else:
                        log.warning(f"Reported IP {ip_address} (from
target port {target_port}), but response format was unexpected or score
missing: {response_json}")

        except ClientResponseError as e:
            response_body = None
            try:
                if hasattr(e, 'history') and e.history:
                     last_response = e.history[-1][1]
                     if last_response: response_body = await
last_response.text()
            except Exception as read_err:
                log.debug(f"Could not read response body from error
history: {read_err}")
                response_body = "(Failed to retrieve error response
body)"
            log.error(
                f"Failed to report IP {ip_address} (from target port
{target_port}): HTTP Error {e.status} - {e.message}. "
                f"URL: {e.request_info.real_url}. "
                f"Response hint: {response_body or '(No body
details)'}"
            )
        except ClientError as e:
            log.error(f"Failed to report IP {ip_address} (from target
port {target_port}): Client/Network Error - {e}")
        except Exception as e:
            log.exception(f"An unexpected error occurred while
reporting IP {ip_address} (from target port {target_port})")
```

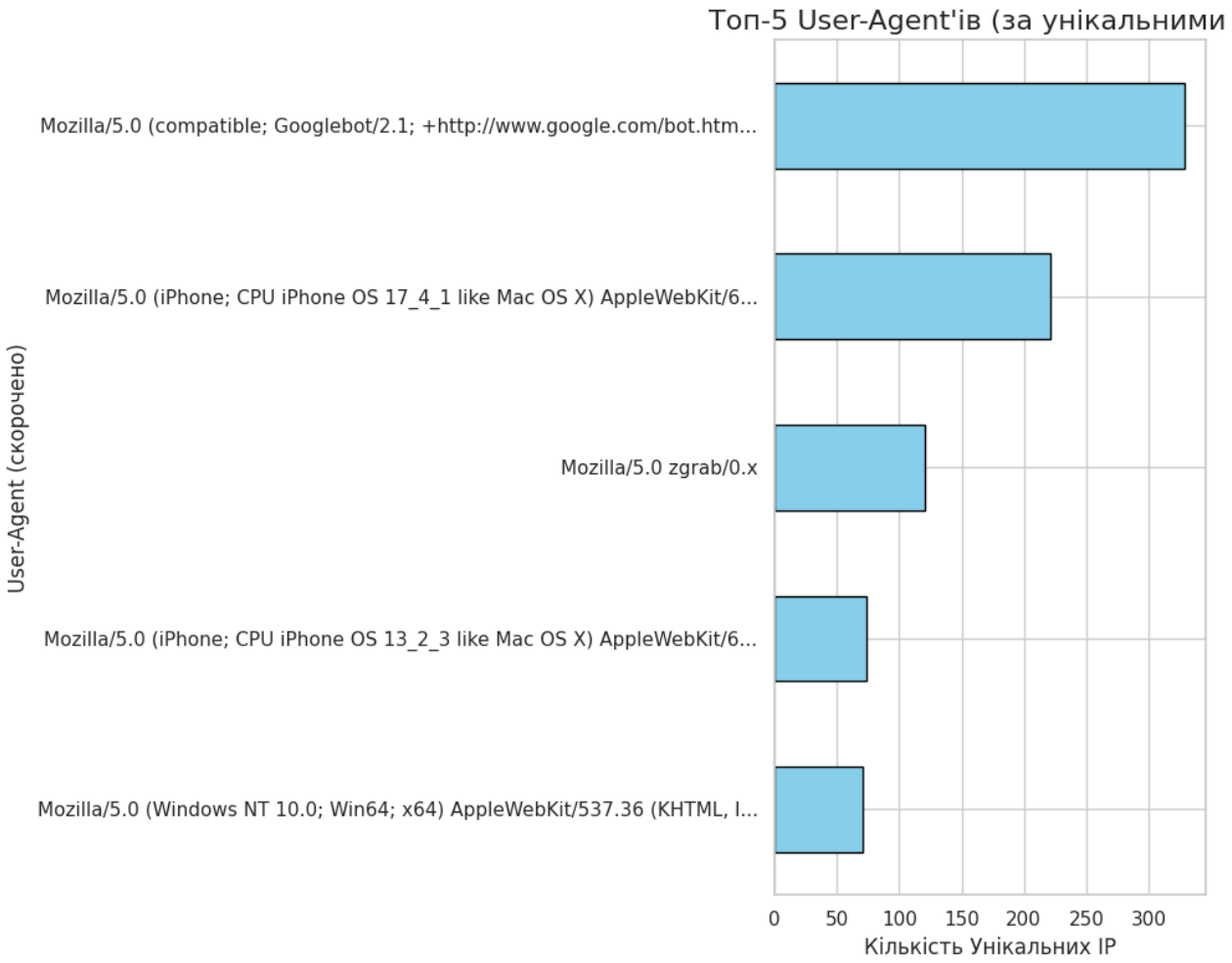
Додаток Б. Візуалізація отриманих результатів



Активність унікальних IP з 07.05 по 21.05



Процентне відношення ботів по країнам



П’ять найчастіших UA(User Agent) по запитам