МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗИГАРЬ ДАНИЛО ДЕНИСОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА СИСТЕМИ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ НА ОСНОВІ МОДИФІКАЦІЇ ПОЛІВ ЗАГОЛОВКІВ СТЕКІВ ПРОТОКОЛІВ TCP/IP

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Наталія ВЕСЕЛОВСЬКА, професор кафедри
інформаційних технологій,
д-р. техн. наук, професор

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2026

АНОТАЦІЯ

Зигарь Д.Д.

Розробка системи прихованої передачі даних на основі модифікації полів заголовків стеків протоколів TCP/IP. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні технології». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

Бакалаврська робота присвячена створенню програмного комплексу для організації прихованого каналу зв'язку в комп'ютерних мережах.

Основна мета дослідження полягає у розробці інструменту, який дозволяє непомітно передавати інформацію шляхом зміни службових полів у заголовках мережевих пакетів, забезпечуючи при цьому цілісність даних.

У процесі роботи проаналізовано будову протоколів TCP/IP, визначено поля, придатні для приховування даних, та досліджено існуючі методи мережевої стеганографії.

В результаті було розроблено систему, що складається з агентів передачі даних (на Python) та веб-інтерфейсу (на Laravel) для керування та моніторингу процесу.

Ключові слова: мережева стеганографія, TCP/IP, прихований канал, Python, Scapy, Laravel, інформаційна безпека.

ABSTRACT

Zyhar D.D.

Development of a system for hidden data transmission based on the modification of header fields of TCP/IP protocol stacks. Specialty 122 “Computer Science”, educational program “Computer Technologies”. Vasyl' Stus Donetsk National University, Vinnytsia, 2026.

The work is devoted to the creation of a software complex for organizing a hidden communication channel in computer networks.

The purpose of the bachelor's thesis is to develop a tool that allows transmitting information unnoticed by modifying service fields in network packet headers, while ensuring data integrity.

During the research, the structure of TCP/IP protocols was analyzed, fields suitable for hiding data were identified, and existing network steganography methods were examined.

As a result, a system consisting of data transmission agents (in Python) and a web interface (in Laravel) for process management and monitoring was developed.

Keywords: network steganography, TCP/IP, covert channel, Python, Scapy, Laravel, information security.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасний етап розвитку цифрових технологій характеризується тотальною інтеграцією комп'ютерних мереж у всі сфери життя - від управління критичною інфраструктурою держави до приватного спілкування. Обсяги даних, що передаються через глобальну мережу Інтернет, зростають у геометричній прогресії, що неминуче призводить до посилення контролю за інформаційними потоками. Урядові структури, інтернет-провайдери та корпорації активно впроваджують системи глибокого аналізу трафіку (DPI - Deep Packet Inspection), які здатні не лише блокувати доступ до певних ресурсів, але й виявляти факт використання захищених каналів зв'язку.

Традиційні методи захисту інформації, такі як шифрування (VPN, TLS, HTTPS), вирішують задачу конфіденційності вмісту, перетворюючи дані на нечитабельний набір символів. Проте вони мають суттєвий недолік: сам факт передачі шифрованих даних є очевидним для стороннього спостерігача. У багатьох сценаріях - наприклад, в умовах суворої цензури, корпоративного шпигунства або моніторингу мережевої активності - саме наявність шифрованого тунелю вже викликає підозру та може стати причиною блокування з'єднання або додаткової перевірки користувача.

У таких умовах виникає нагальна потреба не просто приховати зміст повідомлення, а приховати сам факт його існування. Саме цю задачу вирішує стеганографія - наука про приховану передачу інформації. Особливий інтерес для фахівців з кібербезпеки становить мережева стеганографія, яка використовує особливості будови протоколів передачі даних для організації прихованих каналів (covert channels).

Стек протоколів TCP/IP, на якому базується сучасний Інтернет, був розроблений десятиліття тому і містить низку полів у заголовках пакетів, які є або надлишковими, або варіативними, або зарезервованими для майбутнього використання. Зміна значень цих полів (наприклад, ідентифікатора пакета в IP або

номера послідовності в TCP) дозволяє вбудовувати корисне навантаження без порушення стандартів протоколу. Такий трафік виглядає для систем моніторингу як цілком легітимний, що дозволяє обходити фільтри та передавати дані непомітно.

Попри наявність теоретичних досліджень у цій галузі, практичних інструментів, які б дозволяли зручно організувати такий канал зв'язку, на ринку недостатньо. Тому розробка повноцінної інформаційної системи, яка поєднує низькорівневу роботу з мережею (на Python) та зручний інтерфейс адміністрування (на Laravel), є актуальною науково-практичною задачею, що відповідає сучасним викликам у сфері інформаційної безпеки.

Об'єктом дослідження є процеси прихованого обміну інформацією в комп'ютерних мережах, побудованих на базі стека протоколів TCP/IP.

Предметом дослідження є методи та програмні засоби стеганографічного вбудовування даних у службові поля заголовків мережевих пакетів IPv4 та TCP [3].

Метою роботи є розробка програмного комплексу для організації надійного прихованого каналу передачі даних, стійкого до виявлення стандартними засобами моніторингу мережі, із забезпеченням зручного веб-інтерфейсу керування.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Провести аналіз структури заголовків протоколів TCP/IP та виявити поля, придатні для використання як контейнери для прихованих даних (стегоконтейнери).
2. Дослідити існуючі методи мережевої стеганографії та програмні рішення, виявити їхні недоліки та обмеження.
3. Розробити архітектуру клієнт-серверної системи, яка включає модуль низькорівневої роботи з трафіком (на мові Python) та модуль адміністрування і візуалізації (на фреймворку Laravel).
4. Створити алгоритми розбиття, вбудовування (інжекції) та вилучення (екстракції) даних із мережевих пакетів із забезпеченням контролю цілісності.

5. Виконати програмну реалізацію системи, налаштувати взаємодію між мережевими агентами та центральним сервером керування.
6. Провести тестування розробленого комплексу: перевірити коректність передачі даних, оцінити пропускну здатність прихованого каналу та його непомітність для аналізаторів трафіку (Wireshark) [5].

Методи дослідження. У роботі використано методи системного аналізу - для дослідження структури мережевих протоколів; методи алгоритмізації та програмування - для створення модулів маніпуляції пакетами; методи експериментального дослідження - для перевірки працездатності системи та оцінки її стійкості до виявлення.

Практичне значення одержаних результатів. Розроблений програмний продукт дозволяє здійснювати приховану передачу файлів та текстових повідомлень в умовах обмеженого або контрольованого мережевого доступу. Практична особливість рішення полягає у комбінуванні низькорівневої маніпуляції полями ідентифікації (IP ID) та послідовності (TCP Sequence) з високорівневою системою керування через веб-інтерфейсі. Система може бути використана фахівцями з кібербезпеки для проведення тестів на проникнення (Penetration Testing) з метою перевірки налаштувань систем виявлення вторгнень (IDS/IPS) [10], а також для організації резервних каналів зв’язку.

Структура роботи. Бакалаврська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить [74] сторінок.

1. КОМПЛЕКСНИЙ АНАЛІЗ МЕТОДІВ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ

1.1 Аналіз проблеми захисту інформації в мережах

В умовах глобальної цифровізації та експоненціального зростання обсягів даних, що передаються через публічні мережі, питання забезпечення інформаційної безпеки набуває першочергового значення. Сучасні інформаційно-телекомунікаційні системи функціонують у середовищі з "нульовою довірою" (Zero Trust), де кожен сегмент мережі потенційно може прослуховуватися зловмисниками, корпораціями або державними контролюючими органами.

Класична тріада інформаційної безпеки [10] включає забезпечення конфіденційності, цілісності та доступності даних. Для досягнення конфіденційності традиційно використовуються криптографічні методи захисту. Протоколи, такі як TLS (Transport Layer Security), IPsec (Internet Protocol Security) та SSH (Secure Shell), стали промисловими стандартами для створення захищених тунелів (VPN) [9]. Вони ефективно вирішують задачу захисту *змісту* повідомлення, перетворюючи відкритий текст [9] на псевдовипадкову послідовність бітів (шифротекст), яку неможливо прочитати без відповідного ключа дешифрування.

Однак, в умовах сучасного моніторингу мережевого простору, криптографічного захисту часто виявляється недостатньо. Основна проблема полягає в тому, що криптографія приховує зміст комунікації, але не приховує сам *факт* її наявності. Зашифрований трафік має специфічні ознаки, які легко детектуються засобами аналізу:

1. **Висока ентропія даних.** Зашифровані пакети мають ентропію, близьку до максимальної, що різко відрізняє їх від звичайного текстового або мультимедійного трафіку. Формула ентропії: $H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$. Для ідеального шифротексту AES-128 ймовірність появи кожного байта $P(x_i) \approx \frac{1}{256}$, отже $H(X) \approx 8$ біт на байт.
2. **Специфічні сигнатури протоколів.** Процес встановлення захищеного

з’єднання (Handshake) у протоколах TLS або OpenVPN має чітко визначену структуру, яку легко ідентифікувати.

3. **Метадані.** Навіть при повному шифруванні корисного навантаження, заголовки пакетів (IP-адреси, порти, службові прапори) залишаються відкритими для маршрутизації, що дозволяє аналізувати граф зв’язків користувача ("Traffic Analysis").

Сучасні системи фільтрації контенту та запобігання вторгненням (IDS/IPS) використовують технологію **DPI (Deep Packet Inspection)**. Ця технологія дозволяє аналізувати трафік на рівнях від канального до прикладного (L2–L7 моделі OSI). DPI-системи здатні виявляти використання VPN-тунелів, проксі-серверів або мережі TOR та, залежно від політики безпеки, блокувати такі з’єднання або маркувати їх як "підозрілі" для подальшого ручного аналізу.

У сценаріях, де сам факт використання засобів захисту може скомпрометувати користувача або призвести до блокування каналу зв’язку (наприклад, при обході цензури, захисті від промислового шпигунства або прихованому керуванні віддаленими серверами), виникає необхідність у застосуванні альтернативних методів захисту.

Такою альтернативою є **мережева стеганографія (Network Steganography) [9]**. На відміну від криптографії, метою стеганографії є приховування факту існування повідомлення шляхом його вбудовування в інші, легітимні дані (контейнери). У контексті комп’ютерних мереж контейнерами виступають пакети стандартних протоколів передачі даних, зокрема стека TCP/IP.

Основними перевагами використання стеганографічних каналів (covert channels) є:

- **Непомітність.** Трафік виглядає як стандартний HTTP, DNS або ICMP обмін, що не викликає підозр у систем DPI.
- **Стійкість до блокування.** Оскільки прихований канал паразитує на легітимних протоколах, його блокування часто неможливе без порушення

роботи звичайних сервісів мережі.

Можливість комбінування. Стеганографія не виключає криптографію: приховане повідомлення може бути попередньо зашифрованим, що забезпечує дворівневий захист.

Таким чином, розробка програмних засобів для організації прихованих каналів передачі даних на основі модифікації службових полів мережевих протоколів є актуальним науково-технічним завданням. Це дозволить створити інструментарій для забезпечення конфіденційності в умовах тотального моніторингу, де традиційні криптографічні методи втрачають свою ефективність через легкість їх виявлення.

1.2 Структурний аналіз протоколів TCP/IP та визначення потенційних стегоконтейнерів

Для реалізації прихованого каналу передачі даних необхідно визначити такі елементи мережевих пакетів, модифікація яких не призведе до порушення цілісності з'єднання, відкидання пакета проміжними вузлами (маршрутизаторами, шлюзами) або його некоректної обробки кінцевим отримувачем.

Стек протоколів TCP/IP (Transmission Control Protocol / Internet Protocol) є основою сучасного Інтернету. Його архітектура передбачає інкапсуляцію даних, де кожен рівень додає власний заголовок із службовою інформацією. Аналіз стандартів RFC (Request for Comments), що регламентують ці протоколи, дозволяє виявити поля, які мають властивості надлишковості або невизначеності. Саме ці поля можуть виступати як контейнери для стеганографічного вбудовування.

1.2.1. Аналіз заголовка протоколу IPv4

Протокол IP (Internet Protocol) відповідає за адресацію та маршрутизацію пакетів. Заголовок IPv4 має фіксовану довжину 20 байт (без опцій). Детальний аналіз структури заголовка дозволяє виділити кілька полів, перспективних для використання в якості стегоконтейнерів.

Поле ідентифікації (Identification, IP ID) [3] Це 16-бітне поле (2 байти),

розташоване у другому слові заголовка. Згідно зі стандартом RFC 791 [3], його основне призначення - ідентифікація фрагментів одного й того ж IP-дейтаграми. Якщо пакет перевищує розмір MTU (Maximum Transmission Unit) каналу, він розбивається на фрагменти, і кожен фрагмент отримує однакове значення IP ID, щоб хост-отримувач міг зібрати їх назад.

Однак у сучасних мережах переважна більшість трафіку передається з встановленим прапором **DF (Don’t Fragment)**. Це означає, що фрагментація заборонена, а механізм PMTUD (Path MTU Discovery) автоматично визначає оптимальний розмір пакета. У такому режимі поле Identification втрачає своє критичне значення для збірки фрагментів.

Операційні системи заповнюють це поле по-різному:

- інкрементно (збільшують на 1 для кожного наступного пакета);
- випадковим чином (для підвищення безпеки та захисту від атак на кшталт OS Fingerprinting);
- нульовим значенням (у деяких реалізаціях стека Linux для атомарних пакетів).

Оскільки проміжні маршрутизатори зазвичай не змінюють значення IP ID, це поле є ідеальним кандидатом для вбудовування 16 біт (2 байт) прихованої інформації в кожен пакет. Для зовнішнього спостерігача це виглядатиме як потік пакетів із випадковими ідентифікаторами, що є цілком легітимною поведінкою.

Поле Type of Service (ToS) / Differentiated Services (DS) 8-бітне поле, призначене для забезпечення якості обслуговування (QoS). Історично воно використовувалося для вказівки пріоритету трафіку (наприклад, голосовий трафік має вищий пріоритет, ніж передача файлів).

Сучасна інтерпретація цього поля (RFC 2474) використовує перші 6 біт як DSCP (Differentiated Services Code Point) і останні 2 біти для ECN (Explicit Congestion Notification). Проблема використання цього поля полягає в тому, що багато магістральних маршрутизаторів можуть "обнуляти" або змінювати значення

DSCP згідно з власними політиками QoS. Однак, у межах локальних мереж або певних провайдерів, молодші біти цього поля можуть бути використані для передачі невеликих обсягів службової інформації (наприклад, сигнальних прапорів прихованого протоколу).

Поле Time to Live (TTL) 8-бітне поле, що визначає час життя пакета. Його значення зменшується на одиницю кожним маршрутизатором. Хоча модифікувати його можна, це ненадійно, оскільки кінцеве значення залежить від кількості хопів (hops) у маршруті, який може змінюватися динамічно. Тому TTL доцільно використовувати лише як допоміжний канал (наприклад, парне значення означає "1", непарне - "0"), але не для передачі основного навантаження.

1.2.2. Аналіз заголовка протоколу TCP

Протокол TCP забезпечує надійну доставку потоку байтів. Його заголовок є складнішим за IP і містить поля, необхідні для встановлення з’єднання, керування потоком та підтвердження отримання даних.

Поле Sequence Number (SEQ) та Acknowledgment Number (ACK) Це два 32-бітних поля (по 4 байти кожне), які є критично важливими для роботи TCP.

- **Sequence Number** вказує порядковий номер першого байта даних у сегменті.
- **Acknowledgment Number** містить наступний порядковий номер, який відправник очікує отримати.

Найбільший інтерес для стеганографії становить етап встановлення з’єднання ("потрійне рукостискання" - 3-way handshake). Під час надсилання першого пакета (SYN), клієнт генерує **ISN (Initial Sequence Number)** - початковий номер послідовності. Згідно з вимогами безпеки, ISN повинен бути випадковим, щоб запобігти атакам підміни TCP-сесій (TCP Hijacking).

Ця "випадковість" є ідеальним прикриттям. Ми можемо згенерувати ISN не випадково, а так, щоб у ньому було закодовано 4 байти корисної інформації. Оскільки це 32-бітне число, потенційна ємність одного handshake-пакета є досить високою. Системи аналізу трафіку не можуть достовірно відрізнити "справжній"

випадковий ISN від "псевдовипадкового", що містить дані, без знання секретного ключа або алгоритму генерації.

Зарезервовані біти (Reserved bits) У заголовку TCP, між полем "Data Offset" та прапорами керування, є зарезервована область (раніше 6 біт, зараз, із впровадженням ECN, фактично 3 біти: біти 4, 5, 6 у 13-му та 14-му байтах заголовка). Стандарт вимагає, щоб вони були встановлені в нуль. Встановлення їх в одиницю може бути використане для передачі 3 біт інформації на пакет. Хоча це малий обсяг, такий канал може слугувати для синхронізації або передачі команд керування. Проте, слід враховувати, що деякі суворі брандмауери можуть блокувати пакети з ненульовими зарезервованими бітами.

Опції TCP (TCP Options) Поле опцій має змінну довжину. Однією з найпоширеніших опцій є **Timestamp (мітка часу)**, яка використовується для вимірювання RTT (Round Trip Time). Мітка часу - це монотонно зростаюче число. Молодший біт цього числа має властивості шуму (змінюється швидко і непередбачувано). Використання молодшого значущого біта (LSB - Least Significant Bit) поля Timestamp дозволяє організувати прихований канал за методом LSB-стеганографії, подібним до того, що використовується в аудіо- та графічних файлах.

1.2.3. Висновок щодо вибору стегоконтейнерів

На основі проведеного структурного аналізу можна зробити висновок, що найбільш перспективними для практичної реалізації системи прихованої передачі даних є:

1. **Поле IP Identification (16 біт):** Забезпечує високу пропускну здатність (2 байти на пакет) і має високу стійкість до виявлення в потоці нефрагментованого трафіку.
2. **Поле TCP Initial Sequence Number (32 біти):** Дозволяє передавати значні порції даних (4 байти) на етапі ініціалізації з’єднання, імітуючи стандартну процедуру handshake.

Саме ці два методи будуть покладені в основу алгоритмічної та програмної

реалізації системи, що розробляється в даній роботі. Комбінування цих методів дозволить створити гнучкий інструмент, здатний адаптуватися до різних мережевих умов.

1.3 Огляд існуючих рішень для організації прихованих каналів та їх порівняльний аналіз

Проблема прихованої передачі даних не є новою, і спроби її вирішення здійснювалися дослідниками та хакерами з моменту масового впровадження протоколів TCP/IP. Існує низка програмних продуктів, які реалізують різні методи мережевої стеганографії. Для обґрунтування необхідності розробки нової системи доцільно провести аналіз найбільш відомих рішень, визначити принципи їх роботи, переваги та критичні недоліки.

Аналіз існуючого програмного забезпечення дозволяє класифікувати його за методом вбудовування даних: тунелювання через ICMP/DNS та маніпуляція заголовками TCP/IP.

1.3.1. Засоби тунелювання на основі ICMP та DNS

Найбільш поширеним класом утиліт є ті, що використовують інкапсуляцію корисного навантаження в тіло пакетів службових протоколів.

Ptunnel (Ping Tunnel) Одна з найвідоміших утиліт для організації тунелів через протокол ICMP (Internet Control Message Protocol).

- **Принцип роботи:** Клієнт відправляє ICMP Echo Request (пінг), у "тіло" якого (payload) замість стандартного сміттєвого наповнювача записуються корисні дані. Сервер відповідає ICMP Echo Reply, також містячи дані.
- **Переваги:** Здатність працювати через NAT та прості брандмауери, які дозволяють команду ping.
- **Недоліки:** Високий рівень "шуму". Постійний потік великих ICMP-пакетів є аномалією, яку легко виявляють сучасні системи IDS (Intrusion Detection Systems) на кшталт Snort або Suricata. Крім того, це рішення не є "чистою"

стеганографією, оскільки дані знаходяться у полі навантаження, а не в заголовках.

Iodine (IP over DNS) Інструмент, що дозволяє передавати IPv4-трафік через DNS-запити.

- **Принцип роботи:** Дані кодуються у вигляді довгих субдоменів (наприклад, encoded_data.myserver.com). DNS-сервер декодує їх і відправляє відповідь.
- **Переваги:** Працює навіть у суворо обмежених мережах (наприклад, платні Wi-Fi зони, де доступний лише 53-й порт).
- **Недоліки:** Вкрай низька швидкість передачі та необхідність наявності власного доменного імені. Як і у випадку з Ptunnel, такий трафік легко детектується за аномальною довжиною DNS-запитів та їх частотою.

1.3.2. Засоби маніпуляції заголовками TCP/IP

Цей клас інструментів є більш релевантним до теми роботи, оскільки вони використовують саме службові поля протоколів.

Covert_TCP Класична утиліта, написана мовою C у 1997 році дослідником Крейгом Роулендом (Craig Rowland). Вона стала фактичним стандартом (Proof-of-Concept) для демонстрації можливостей маніпуляції заголовками.

Принцип роботи: Дозволяє вбудовувати по одному байту даних у поля IP Identification або TCP Sequence Number.

Переваги:

- Висока скритність порівняно з тунелюванням. Пакети виглядають як порожні (або майже порожні) TCP-сегменти.

Недоліки:

- **Застарілість коду:** Утиліта не оновлювалася десятиліттями і складно компілюється на сучасних 64-бітних системах.
- **Відсутність шифрування:** Дані передаються у відкритому вигляді (ASCII коди в заголовках), що дозволяє легко відновити повідомлення при перехопленні.

- **Консольний інтерфейс:** Керування здійснюється виключно через командний рядок, що ускладнює моніторинг процесу передачі великих файлів.

Nmap (Idle Scan / Zombie Scan) Хоча Nmap є сканером безпеки, а не засобом передачі даних, він використовує механізм передбачення IP ID для проведення прихованого сканування портів ("Idle Scan").

Значення для дослідження: Успішне функціонування цього методу в глобальній мережі доводить практичну можливість використання поля IP ID як каналу витоку інформації або каналу керування. Це підтверджує правильність вибору поля IP Identification як одного з основних стегоконтейнерів у даній роботі.

1.3.3. Аналіз недоліків існуючих рішень та формування вимог до нової системи

Проведений аналіз дозволяє виділити загальні проблеми існуючого програмного забезпечення для прихованої передачі даних:

1. **Відсутність централізованого керування.** Більшість інструментів (Covert_TCP, Ptunnel) працюють за схемою "точка-точка" через консоль. Відсутня можливість візуалізувати процес, переглядати логи в зручному форматі або керувати декількома агентами одночасно.
2. **Низька адаптивність.** Існуючі утиліти зазвичай жорстко прив'язані до одного методу (тільки IP ID або тільки TCP Seq). Немає можливості динамічно перемикати метод вбудовування залежно від умов мережі.
3. **Відсутність інтеграції з сучасними веб-технологіями.** Жодне з розглянутих рішень не пропонує REST API або веб-інтерфейсу для інтеграції з системами моніторингу, що є стандартом для сучасних Enterprise-рішень.
4. **Складність розгортання.** Більшість інструментів потребують специфічних залежностей або компіляції вихідного коду, що ускладнює їх швидке використання.

1.4 Висновки

У першому розділі роботи проведено комплексний аналіз проблеми забезпечення прихованої передачі даних в умовах сучасного інформаційного простору, що характеризується тотальним моніторингом мережевого трафіку.

За результатами дослідження можна зробити наступні висновки:

Обмеженість традиційних методів захисту. Встановлено, що класичні криптографічні засоби (VPN, TLS), хоча й забезпечують конфіденційність змісту повідомлень, не здатні приховати факт їх передачі. Використання технологій глибокого аналізу пакетів (DPI) дозволяє детектувати та блокувати шифровані канали, що обумовлює необхідність застосування методів мережевої стеганографії.

Потенціал протоколів TCP/IP. Детальний структурний аналіз заголовків протоколів IPv4 та TCP виявив наявність полів, які мають властивості надлишковості або варіативності. Визначено, що найбільш перспективними контейнерами для вбудовування прихованих даних є поле ідентифікації (IP Identification, 16 біт) та поле порядкового номера (TCP Sequence Number, 32 біти). Їх модифікація при правильній реалізації не порушує цілісність з'єднання і дозволяє імітувати легітимний трафік.

Недоліки існуючих рішень. Критичний огляд існуючого програмного забезпечення (Covert_TCP, Ptunnel та інших) показав, що більшість доступних інструментів є морально застарілими, мають обмежений функціонал (працюють лише в консольному режимі), не підтримують сучасні алгоритми шифрування та позбавлені зручних засобів централізованого керування. Це робить їх малопридатними для використання в сучасних динамічних мережевих середовищах.

Обґрунтування розробки. На основі виявлених проблем сформовано концепцію створення нового програмного комплексу. Доведено доцільність використання дворівневої архітектури: низькорівневого модуля на мові Python (з використанням бібліотеки Scapy) для безпосередньої маніпуляції пакетами та

високорівневого веб-інтерфейсу на базі фреймворку Laravel для адміністрування, візуалізації логів та керування агентами. Такий підхід дозволить поєднати ефективність стеганографічних алгоритмів із зручністю сучасних веб-технологій.

Таким чином, результати першого розділу створюють теоретичне підґрунтя для переходу до етапу проектування архітектури системи та розробки алгоритмів її функціонування, що буде розглянуто в наступних розділах роботи.

2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА АЛГОРИТМІВ СИСТЕМИ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ

2.1 Формування вимог та концептуальне моделювання архітектури

Етап формування вимог є фундаментом для створення якісного програмного продукту, оскільки він визначає функціональні межі системи та критерії її успішної роботи. Специфіка розробки засобів мережевої стеганографії суттєво відрізняється від створення типових веб-додатків. Якщо для комерційних систем пріоритетними є швидкість обробки запитів та зручність інтерфейсу, то для засобів прихованої передачі даних на перший план виходять непомітність функціонування та стійкість до виявлення засобами моніторингу.

Проектування системи базується на необхідності реалізації розподіленої архітектури, де низькорівневі операції з мережевим трафіком виконуються автономними агентами, а керування та моніторинг здійснюються централізовано. Функціональне ядро системи повинно забезпечувати повний цикл обробки даних: від зчитування цільового файлу до його відновлення на стороні отримувача. Ключовою вимогою до модуля відправника є здатність автоматично фрагментувати довільні дані відповідно до ємності обраного стегоконтейнера (поля IP Identification або TCP Sequence Number) та коректно розраховувати контрольні суми пакетів, щоб уникнути їх відкидання маршрутизаторами. Модуль отримувача, у свою чергу, повинен працювати в режимі прослуховування мережевого інтерфейсу, фільтруючи вхідний трафік за заданими сигнатурами та збираючи отримані фрагменти в цілісний файл.

Окремий блок вимог стосується підсистеми керування, яка реалізується у вигляді веб-інтерфейсу. Вона має надавати адміністратору інструменти для моніторингу статусу агентів у реальному часі, візуалізації логів передачі та керування налаштуваннями протоколу. Важливою умовою є здатність системи зберігати історію транзакцій, включаючи час відправлення, обсяг переданих даних та ідентифікатори вузлів мережі.

Серед нефункціональних вимог критичне значення має забезпечення скритності комунікації. Генеровані пакети повинні повністю відповідати стандартам RFC, а статистичні характеристики потоку даних не мають створювати аномалій, що фіксуються системами виявлення вторгнень (IDS) [10]. Також система повинна гарантувати цілісність даних в умовах ненадійного середовища передачі, що вимагає впровадження механізмів хешування та перевірки контрольних сум для кожного повідомлення. Враховуючи гетерогенність сучасних мереж, програмні агенти повинні бути кросплатформними та забезпечувати стабільну роботу як в середовищі Linux, так і Windows, не створюючи при цьому критичного навантаження на обчислювальні ресурси хост-системи.

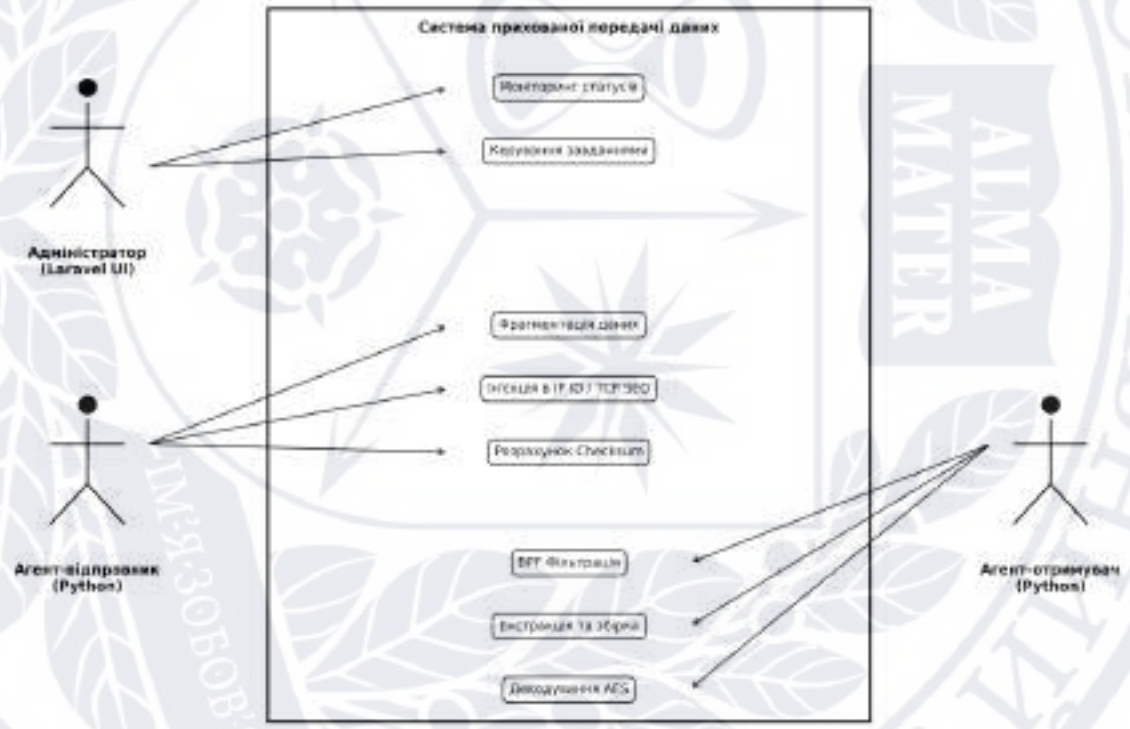

Рисунок 2.1. Діаграма прецедентів системи прихованої передачі даних.

Представлена діаграма (рис. 2.1) ілюструє чіткий розподіл функціональних обов’язків. Адміністратор взаємодіє із системою виключно через високорівневий інтерфейс керування завданнями та моніторингу. В той же час, мережеві агенти автономно реалізують критичні низькорівневі процеси: від маніпуляції

контрольними сумами до криптографічного декодування, що забезпечує мінімальну затримку при обробці трафіку.

Проектування архітектури програмного забезпечення для організації прихованих каналів передачі даних вимагає застосування гібридного підходу, що поєднує принципи побудови розподілених клієнт-серверних систем та низькорівневої взаємодії з мережевим обладнанням. Загальна структура розроблюваного комплексу базується на модульній архітектурі, яка дозволяє чітко розмежувати логіку маніпуляції пакетами, процеси зберігання даних та інтерфейси взаємодії з користувачем. Такий підхід забезпечує гнучкість налаштування, масштабованість системи та спрощує процес налагодження окремих компонентів.

Основу архітектури складають три ключові компоненти, що взаємодіють між собою за чітко визначеними протоколами. Першим компонентом є підсистема мережевих агентів, яка реалізується мовою програмування Python [2]. Вибір цієї мови зумовлений наявністю потужних бібліотек для роботи з мережевим стеком, зокрема Scapy, що дозволяє формувати довільні мережеві пакети в обхід стандартних механізмів операційної системи. Агенти поділяються на два функціональні типи: агент-відправник (Sender), який відповідає за фрагментацію файлів та їх ін’єкцію в службові поля заголовків IPv4/TCP, та агент-отримувач (Receiver), що здійснює прослуховування трафіку (sniffing), фільтрацію пакетів за сигнатурами та екстракцію прихованих даних.

Другим компонентом є центральний сервер керування та моніторингу (Command & Control Server), побудований на базі PHP-фреймворку Laravel [8]. Використання архітектурного паттерну MVC (Model-View-Controller), притаманного цьому фреймворку, дозволяє ефективно організувати логіку обробки запитів від агентів та відображення даних. Серверна частина відповідає за автентифікацію користувачів, генерацію конфігурацій для агентів (наприклад, вибір методу стеганографії або цільових IP-адрес) та обробку отриманих логів. Взаємодія між Python-агентами та Laravel-бекендом здійснюється через REST API,

що дозволяє передавати статус виконання завдань та статистику в форматі JSON.

Третім компонентом є підсистема зберігання даних, реалізована на основі реляційної системи керування базами даних MySQL (або MariaDB). База даних виступає єдиним джерелом істини для системи, зберігаючи облікові записи адміністраторів, історію транзакцій, конфігураційні параметри та, за необхідності, метадані переданих файлів. Використання ORM (Object-Relational Mapping) Eloquent у Laravel забезпечує абстрагування від прямих SQL-запитів, підвищуючи безпеку та спрощуючи підтримку коду.

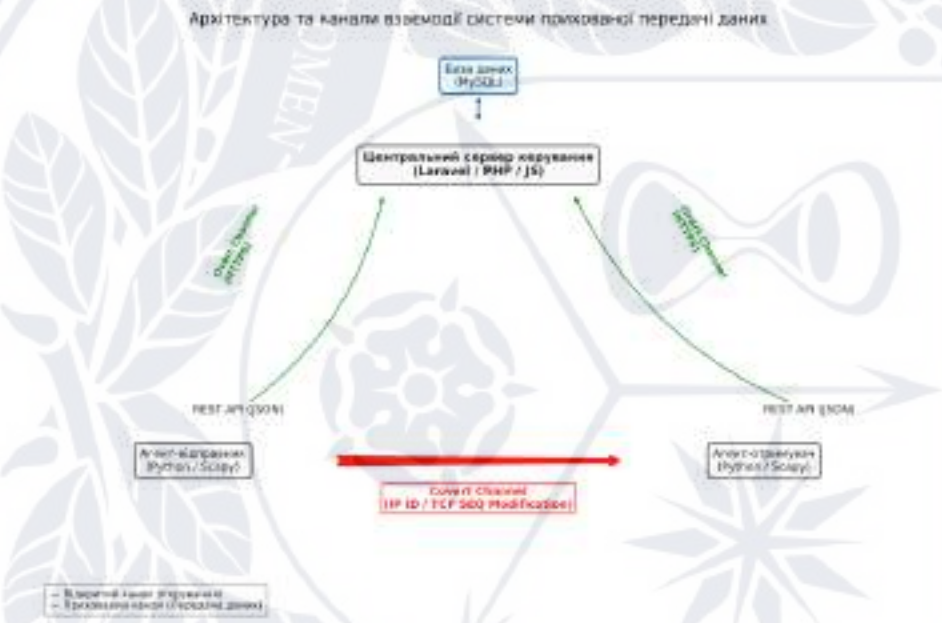

Рисунок 2.2. Концептуальна схема архітектури та інформаційних потоків.

Представлена архітектурна схема (рис. 2.2) демонструє дворівневу модель комунікації. Вертикальні зв'язки (Overt Channel) забезпечують стабільність системи: через них агенти отримують завдання та звітують про стан. Горизонтальний зв'язок (Covert Channel) реалізує основну мету роботи - передачу даних безпосередньо між вузлами мережі, минаючи центральний сервер, що мінімізує обсяг потенційно підозрілого трафіку в межах одного сегмента.

Клієнтська частина веб-інтерфейсу (Frontend) реалізується з використанням JavaScript, що забезпечує динамічне оновлення інформації на сторінці без необхідності її перезавантаження. Це критично важливо для моніторингу процесу

передачі даних у реальному часі, дозволяючи адміністратору миттєво бачити статус доставки кожного фрагмента повідомлення. Така архітектура гарантує, що високорівнева логіка керування не впливає на продуктивність низькорівневих процесів обробки пакетів, забезпечуючи стабільність та надійність функціонування всього комплексу.

2.2 Проектування структури бази даних та веб-інтерфейсу керування

Ефективне функціонування системи прихованої передачі даних неможливе без надійної підсистеми зберігання інформації та зручних засобів взаємодії з оператором. Проектування бази даних (БД) та інтерфейсу користувача (UI) є взаємопов'язаними процесами, оскільки структура даних визначає можливості інтерфейсу, а вимоги до зручності керування диктують необхідність зберігання певних атрибутів системи.

2.2.1. Інфологічне моделювання бази даних

Для забезпечення централізованого керування мережевими агентами та логування процесів передачі даних обрано реляційну модель бази даних, яка буде реалізована за допомогою СУБД MySQL. Вибір цієї системи зумовлений її високою продуктивністю, безкоштовністю та нативною підтримкою у фреймворку Laravel.

При проектуванні схеми бази даних було дотримано принципів нормалізації (до третьої нормальної форми включно) для уникнення надлишковості даних та забезпечення їх цілісності. Основними сутностями системи є:

1. **Сутність Users (Користувачі).** Зберігає дані про адміністраторів системи, які мають доступ до панелі керування.
   - *Ключові атрибути:* id (PK), name, email (логін), password (хеш), role (рівень доступу: admin/viewer).
2. **Сутність Agents (Мережеві агенти).** Описує програмні модулі (Python-скрипти), розгорнуті на віддалених вузлах мережі. Ця таблиця дозволяє системі розрізняти відправників та отримувачів.

- *Ключові атрибути:*
  - id (PK);
  - uuid (унікальний ідентифікатор агента, що генерується при першому запуску);
  - ip_address (поточна IP-адреса вузла);
  - status (enum: 'online', 'offline', 'busy');
  - last_heartbeat (timestamp останнього відгуку).

3. **Сутність Transmissions (Сеанси передачі).** Фіксує кожен факт передачі даних (файлу або повідомлення). Це центральна таблиця для логування активності.
   - *Ключові атрибути:*
     - id (PK);
     - sender_agent_id (FK -> Agents);
     - receiver_agent_id (FK -> Agents);
     - file_name (назва переданого файлу);
     - file_hash (контрольна сума MD5/SHA256 для перевірки цілісності);
     - stego_method (enum: 'ip_id', 'tcp_seq');
     - status (enum: 'pending', 'processing', 'completed', 'failed');
     - started_at, completed_at.
4. **Сутність Logs (Детальні логи).** Зберігає технічну інформацію про роботу системи, помилки та налагоджувальні дані.
   - *Ключові атрибути:*
     - id, agent_id та message;
     - level (info/error/warning);
     - created_at.

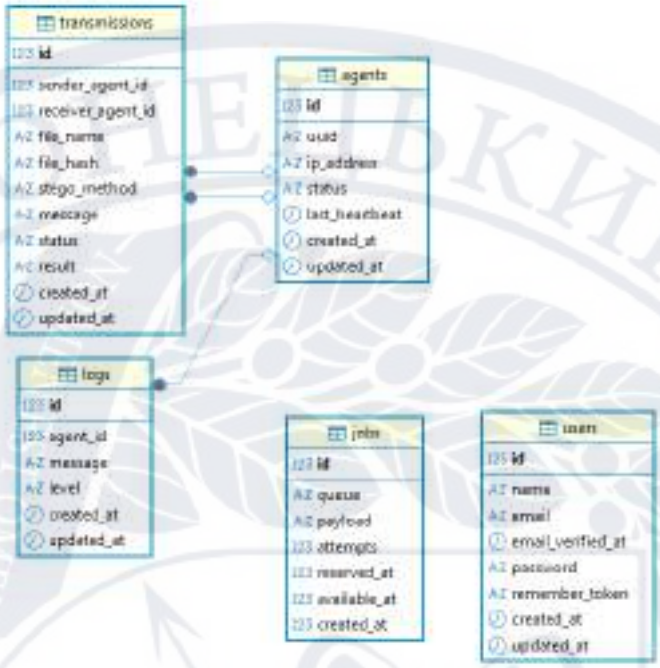

Рисунок 2.3. Відображення зв’язків таблиць через DBeaver.

На рисунку 2.3 представлена ER-діаграма (Entity-Relationship Diagram) через застосунок dbeaver (для перегляду та роботи з різними БД), що відображає зв’язки між описаними сутностями. Використання зовнішніх ключів (Foreign Keys) гарантує каскадну цілісність даних: наприклад, при видаленні агента історія його передач може бути збережена або архівована.

2.2.2. Проектування веб-інтерфейсу керування (Dashboard)

Веб-інтерфейс є "обличчям" системи для адміністратора. Він реалізується на базі шаблонізатора Laravel Blade [8] з використанням компонентів JavaScript (Vue.js або "ванільний" JS) для забезпечення асинхронної взаємодії з сервером (AJAX). Це дозволяє оновлювати статус передачі даних без перезавантаження сторінки.

Структура інтерфейсу включає наступні основні розділи:

1. **Головна панель (Dashboard).** Відображає зведену статистику: кількість активних агентів, графік активності передач за останню добу, список останніх подій. Тут розміщуються віджети стану системи ("Health Check").

2. **Менеджер агентів (Agents Management).** Таблиця зі списком усіх зареєстрованих вузлів.

   - *Функціонал:* Адміністратор бачить, які агенти зараз "online". Реалізовано кольорову індикацію статусу (зелений - активний, червоний - недоступний).

3. **Центр керування передачею (Transmission Control).** Форма для ініціювання нового сеансу прихованого зв'язку.

   - *Елементи керування:* Випадаючі списки для вибору "Відправника" та "Отримувача", поле завантаження файлу (File Input), перемикач методу стеганографії (IP ID / TCP SEQ).

   - *Логіка:* Після натискання кнопки "Start", сервер створює запис у таблиці Transmissions і через API відправляє команду відповідному Python-агенту.

4. **Журнал аудиту (Audit Log).** Деталізований список усіх дій в системі. Дозволяє фільтрувати записи за датою, типом події або конкретним агентом. Це критично важливо для аналізу інцидентів та налагодження роботи алгоритмів.

Рисунок 2.3.1 Макет головної сторінки панелі керування.

На рисунку 2.3.1 наведено макет (Wireframe) головної сторінки панелі керування. Розроблений інтерфейс спроектовано з урахуванням принципів юзабіліті (Usability) та мінімалізму, щоб оператор міг швидко приймати рішення в умовах дефіциту часу.

Використання фреймворку CSS (наприклад, Bootstrap або Tailwind) забезпечує адаптивність інтерфейсу, що дозволяє керувати системою не лише з ПК, але й з мобільних пристроїв, що підвищує оперативність реагування на події.

2.3. Розробка алгоритмів вбудовування та вилучення даних

Процес приховування інформації в мережевому трафіку (Embedding) є ключовою функцією розроблюваної системи. Основна складність цього процесу полягає не лише у записі даних у відповідні поля, але й у збереженні валідності

мережевих пакетів з точки зору стандартів протоколів TCP/IP. Некоректно сформований пакет буде відкинутий першим же маршрутизатором на шляху до отримувача, що призведе до втрати прихованого повідомлення та демаскування факту аномальної мережевої активності.

Розроблений алгоритм вбудовування складається з чотирьох послідовних етапів: підготовка корисного навантаження, фрагментація, ін'єкція даних та перерахунок контрольних сум [7].

Етап 1. Підготовка корисного навантаження (Payload Preparation)

На вхід алгоритму надходить файл або текстове повідомлення, ініційоване оператором через веб-інтерфейс (Laravel). Оскільки мережеві протоколи оперують байтами, вхідні дані незалежно від їхнього формату перетворюються на одновимірний масив байтів $M=\{m1,m2,\ldots,mk\}$, де k - загальний розмір повідомлення у байтах.

Для забезпечення додаткового рівня безпеки перед вбудовуванням масив M може бути зашифрований симетричним алгоритмом (наприклад, AES-256), що гарантує неможливість прочитання даних навіть у випадку виявлення стеганографічного каналу.

Етап 2. Фрагментація (Chunking)

Оскільки ємність обраних стегоконтейнерів обмежена, масив M необхідно розбити на фрагменти F_i, розмір яких точно відповідає розміру контейнера.

Нехай C - пропускна здатність контейнера в байтах на один пакет.

- Для методу IP Identification C = 2 байти (16 біт).
- Для методу TCP Sequence Number C = 4 байти (32 біти).

Алгоритм розбиває масив M на N фрагментів, де $N=\lceil C/k \rceil$. Якщо розмір останнього фрагмента менший за C, він доповнюється нульовими байтами (Padding) до необхідної довжини.

Етап 3. Ін'єкція даних (Injection)

На цьому етапі агент-відправник (реалізований на мові Python) починає

генерацію транзитного трафіку або перехоплення легітимних пакетів для їх модифікації. Процес ін'єкції полягає у перезаписі цільового поля заголовка значенням фрагмента F_i.

Математично це можна описати як операцію присвоювання:

- Packet[IP].ID←F_i (для IP ID)
- Packet[TCP].SEQ←F (для TCP SEQ)

Важливо зазначити, що генерація "чистих" пакетів лише з прихованими даними може виглядати підозріло. Тому алгоритм передбачає додавання випадкового "сміттєвого" корисного навантаження (Payload) у сам TCP-сегмент або ICMP-запит, щоб пакет імітував реальну взаємодію клієнта з сервером.

Етап 4. Перерахунок контрольних сум (Checksum Recalculation) [7]

Це найбільш критичний етап алгоритму. Будь-яка зміна заголовка пакета (включаючи поля IP ID або TCP SEQ) порушує його контрольну суму. Маршрутизатори перевіряють контрольну суму IP-заголовка на кожному хопі, і якщо вона не збігається, пакет знищується (Drop).

Контрольна сума в протоколах IPv4 та TCP розраховується як 16-бітне зворотне доповнення [7] (1's complement) суми всіх 16-бітних слів заголовка. Алгоритм перерахунку виглядає наступним чином:

1. Поле поточної контрольної суми (Checksum) встановлюється у нуль: CS = 0.
2. Заголовок пакета розбивається на послідовність 16-бітних слів W_1, W_2, ..., W_n.
3. Обчислюється сума всіх слів:

$$S = \sum_{j=1}^{n} W_j$$

4. Якщо під час додавання виникає переповнення (сума перевищує 16 біт, тобто S > 0xFFFF), старші біти (carry) переносяться і додаються до молодших біт: S=(SAND0xFFFF)+(S≫16)
5. Фінальна контрольна сума CS є інверсією отриманої суми CS=∼S(побітове

HI)

У розроблюваній системі ця математична операція абстрагується завдяки використанню бібліотеки Scapy, яка автоматично виконує перерахунок контрольних сум перед відправкою сформованого пакета в мережевий інтерфейс. Однак розуміння цього апарату є обов'язковим для проектування стійкого прихованого каналу.

Після завершення перерахунку, модифікований пакет, що містить фрагмент прихованих даних F_i, передається на рівень абстракції мережевого інтерфейсу (Network Interface Card) для фізичної відправки отримувачу. Процес повторюється ітеративно до повного вичерпання масиву даних M.

Процес вилучення (Extraction) прихованої інформації є зворотним до процесу вбудовування і виконується на стороні агента-отримувача. Складність цього етапу полягає у необхідності виокремлення цільових пакетів із загального потоку мережевого трафіку ("шуму") в режимі реального часу, без створення затримок у роботі хост-системи.

Агент-отримувач, реалізований мовою Python, переводить мережевий інтерфейс у так званий "змішаний режим" (promiscuous mode) або використовує механізми "сирих сокетів" [2] (Raw Sockets) для доступу до пакетів на канальному рівні (L2 моделі OSI). Логіка роботи модуля вилучення розбивається на наступні етапи:

Етап 1. Захоплення та попередня фільтрація трафіку (Sniffing & BPF Filtering)

Для оптимізації використання ресурсів процесора агент не повинен аналізувати кожен пакет у мережі. Замість цього на рівні ядра операційної системи застосовується синтаксис BPF (Berkeley Packet Filter). Фільтр налаштовується таким чином, щоб пропускати до простору користувача (user space) лише пакети, що відповідають заданим критеріям.

Наприклад, якщо прихований канал базується на протоколі TCP, фільтр може мати вигляд: tcp and src host <IP_відправника> and dst port <Цільовий_порт>. Це відсікає

до 99% нерелевантного трафіку ще до початку програмної обробки алгоритмом.

Етап 2. Синхронізація та ідентифікація стего-пакетів

Навіть відфільтрований трафік від цільового IP містить легітимні службові пакети, які не несуть прихованого навантаження. Агент-отримувач повинен чітко розпізнати початок передачі повідомлення (Start of Transmission, SoT) та її завершення (End of Transmission, EoT).

Для цього використовується механізм "сигнатур" або специфічної послідовності прапорів. Наприклад, домовленість між агентами може полягати в тому, що сеанс передачі завжди починається з TCP-пакета з одночасно встановленими прапорами PSH та URG (що є рідкістю у звичайному трафіку) і певним "магічним числом" у полі Sequence Number, яке слугує ключем сеансу.

Етап 3. Екстракція даних (Extraction)

Після фіксації початку сеансу агент починає послідовне зчитування значень із заздалегідь визначених стегоконтейнерів. Залежно від обраного методу, алгоритм виконує операцію побітового зсуву та маскування (Bitwise AND) для вилучення корисного навантаження:

$F_i = Packet[IP].ID \text{ AND } 0xFFFF$ (для 16-бітного контейнера IP ID)

$F_i = Packet[TCP].SEQ \text{ AND } 0xFFFFFFFF$ (для 32-бітного контейнера TCP SEQ)

Вилучені фрагменти F_i послідовно записуються до тимчасового буфера в оперативній пам'яті $B=\{F1,F2,\ldots,Fn\}$

Етап 4. Відновлення послідовності та обробка втрат (Reassembly)

Особливістю IP-мереж є відсутність гарантії доставки пакетів у правильному порядку, а також ймовірність втрати частини трафіку (Packet Loss). Оскільки приховані дані "їдуть" верхи на цих пакетах, вони успадковують ці проблеми.

Якщо прихований канал використовує поле IP ID, яке є атомарним і не має власних механізмів послідовності, агент-відправник повинен резервувати частину самого контейнера під порядковий номер. Наприклад, з 16 біт IP ID: 4 біти виділяються під номер послідовності фрагмента (Sequence Tag), а 12 біт - під самі дані.

Агент-отримувач зчитує Sequence Tag, сортує буфер B відповідно до цих номерів і перевіряє наявність "дірок" (пропущених фрагментів). У разі втрати, через зворотний канал керування (Overt Channel) серверу надсилається запит на повторну передачу втраченого фрагмента (NACK - Negative Acknowledgment).

Етап 5. Декодування та фіналізація

Після отримання сигналу EoT та успішної перевірки цілісності зібраного буфера, масив байтів B піддається зворотному перетворенню. Якщо дані були зашифровані на етапі вбудовування, застосовується симетричний ключ для дешифрування. Далі байти конвертуються у початковий формат (зберігаються на жорсткий диск як файл або передаються до бази даних як текстове повідомлення).

Завершальним кроком цього алгоритму є формування звіту про успішне отримання даних, який агент відправляє на центральний сервер (Laravel) через REST API, після чого переходить назад у режим очікування (Listening state).

2.4 Розробка схеми взаємодії мережевих агентів із веб-сервером

Ефективне функціонування розподіленої системи прихованої передачі даних вимагає наявності надійного та захищеного каналу керування (Command and Control, C2). У той час як передача корисного навантаження здійснюється через прихований канал (Covert Channel) шляхом маніпуляції заголовками TCP/IP, координація цього процесу, обмін конфігураціями та збір статистики відбуваються через відкритий службовий канал (Overt Channel).

Основою для реалізації відкритого каналу в даній роботі обрано архітектурний стиль REST (Representational State Transfer). Взаємодія між мережевими агентами, написаними мовою Python, та центральним сервером на базі фреймворку Laravel здійснюється за протоколом HTTP/HTTPS. Форматом обміну даними (серіалізації) виступає JSON (JavaScript Object Notation), що забезпечує легкість парсингу як на стороні клієнта, так і на стороні сервера.

2.4.1. Автентифікація та безпека відкритого каналу

Оскільки система керує процесами мережевої взаємодії, критично важливим

є захист REST API від несанкціонованого доступу. Для автентифікації агентів використовується механізм статичних API-ключів (Bearer Tokens). Під час першого розгортання кожен Python-агент отримує унікальний криптографічний токен. Усі HTTP-запити від агента до сервера містять цей токен у заголовку Authorization. На стороні Laravel реалізовано проміжне програмне забезпечення (Middleware), яке перевіряє валідність токена перед передачею запиту до контролера. Крім того, весь службовий трафік має інкапсулюватися в TLS-тунель (HTTPS), щоб унеможливити перехоплення команд керування системами DPI.

2.4.2. Життєвий цикл взаємодії (Lifecycle)

Логіку взаємодії можна розділити на чотири основні фази, кожна з яких обслуговується відповідними кінцевими точками (Endpoints) на сервері:

Фаза 1. Ініціалізація та моніторинг стану (Heartbeat) Після запуску агент повинен повідомити сервер про свою готовність до роботи. Для цього використовується механізм "серцебиття" (Heartbeat).

- Агент періодично (наприклад, кожні 30 секунд) відправляє POST-запит на маршрут /api/v1/agents/heartbeat.
- У тілі запиту передається поточна IP-адреса вузла, завантаженість процесора та статус (вільний/зайнятий).
- Laravel оновлює запис у таблиці Agents, фіксуючи час останнього відгуку (last_heartbeat). Якщо агент не надсилає Heartbeat більше визначеного таймауту, веб-інтерфейс позначає його як "Offline".

Фаза 2. Отримання завдань (Task Polling) Оскільки агенти знаходяться за NAT або брандмауерами, сервер не може напряму ініціювати з'єднання з ними (відсутня біла IP-адреса). Тому використовується модель опитування (Polling).

- Агент-відправник періодично звертається до маршруту /api/v1/tasks/pending.
- Якщо адміністратор через веб-панель ініціював нову передачу, сервер повертає JSON-об'єкт із конфігурацією завдання: цільова IP-адреса отримувача, ідентифікатор цільового агента, метод стеганографії (IP ID або

TCP SEQ) та власне корисне навантаження (у вигляді Base64-кодованого рядка).

Фаза 3. Синхронізація сеансу зв'язку Для успішної передачі агент-отримувач повинен знати, що йому слід перейти в режим прослуховування (Sniffing mode).

- Сервер надсилає команду агенту-отримувачу на підготовку до прийому даних (очікування пакетів з певної IP-адреси).
- Лише після підтвердження готовності від отримувача (запит на /api/v1/tasks/ready), сервер віддає команду агенту-відправнику на початок ін'єкції даних у мережевий трафік.

Фаза 4. Логування та передача результатів Під час роботи прихованого каналу агенти повинні звітувати про прогрес.

- Відправник асинхронно відправляє статистику (кількість згенерованих пакетів, відсоток виконання) на маршрут /api/v1/tasks/{id}/progress.
- Отримувач, після фіксації сигналу завершення передачі (End of Transmission) та збірки файлу, відправляє POST-запит на /api/v1/tasks/{id}/complete. Тіло цього запиту містить вилучені дані та розраховану контрольну суму для перевірки цілісності.
- Сервер Laravel зберігає отримані дані в базу даних MySQL, змінює статус завдання на "Completed" та через механізм WebSockets (наприклад, Laravel Reverb або Pusher) миттєво оновлює веб-інтерфейс адміністратора.

Розроблена схема взаємодії чітко розмежовує обов'язки компонентів системи. Використання REST API забезпечує слабку зв'язність (loose coupling) між ядром на Python та веб-панеллю на PHP, що робить систему стійкою до збоїв та легко масштабованою.

2.5. Розрахунок пропускної здатності та надійності прихованого каналу

Для комплексного обґрунтування ефективності розробленої системи прихованої передачі даних необхідно побудувати математичну модель, яка

дозволить кількісно оцінити ключові параметри стеганографічного каналу. До таких параметрів належать теоретична пропускна здатність, ефективна швидкість передачі з урахуванням втрат у мережі, а також загальний час, необхідний для доставки повідомлення заданого обсягу.Основою для розрахунку теоретичної пропускної здатності (ємності) прихованого каналу є розмір обраного стегоконтейнера та інтенсивність генерації мережевого трафіку. Позначимо загальний обсяг контейнера в бітах як V. Для методу, що використовує поле ідентифікації протоколу IPv4 (IP ID), цей показник становить 16 біт. Для методу на основі початкового номера послідовності протоколу TCP (Initial Sequence Number) обсяг дорівнює 32 бітам. Однак, як було визначено під час проектування алгоритмів, частина цього обсягу повинна резервуватися для службової інформації, зокрема для ідентифікатора послідовності фрагмента, який необхідний для правильного збирання файлу на стороні отримувача. Якщо позначити розмір службової частини в бітах як S, то корисний обсяг даних в одному пакеті складатиме різницю V - S.

Нехай R позначає швидкість генерації пакетів агентом-відправником, виражену в пакетах за секунду (Packet Rate). Тоді теоретична пропускна здатність прихованого каналу C_{theor}, виражена в бітах за секунду, розраховується як добуток кількості пакетів на корисний обсяг даних в одному пакеті. Відповідна математична залежність має вигляд $C_{theor} = R * (V - S)$. З цієї формули випливає, що збільшення швидкості відправки пакетів прямо пропорційно підвищує пропускну здатність.

Проте на практиці інтенсивність R жорстко обмежується вимогами щодо скритності: занадто велика кількість пакетів, згенерованих за коротку одиницю часу, створить аномалію в статистичному профілі мережі, що буде негайно зафіксовано системами виявлення вторгнень.Реальні комп'ютерні мережі функціонують в умовах постійних перешкод, перевантажень маршрутизаторів та колізій, що неминуче призводить до втрати частини пакетів. Оскільки прихований канал базується на маніпуляції заголовками транзитних пакетів, він повністю

успадковує ці фізичні обмеження середовища передачі. Для врахування цього фактору введемо параметр p, який позначає ймовірність втрати або пошкодження одного пакета під час його проходження від відправника до отримувача, де значення p лежить у діапазоні від 0 до 1. У разі втрати фрагмента система ініціює процедуру його повторної передачі через механізм негативних підтверджень, що генеруються на рівні відкритого каналу керування.

З урахуванням ймовірності втрат p, ефективна пропускна здатність системи C_{eff} буде нижчою за теоретичну і визначатиметься формулою $C_{eff} = C_{theor} * (1 - p)$.

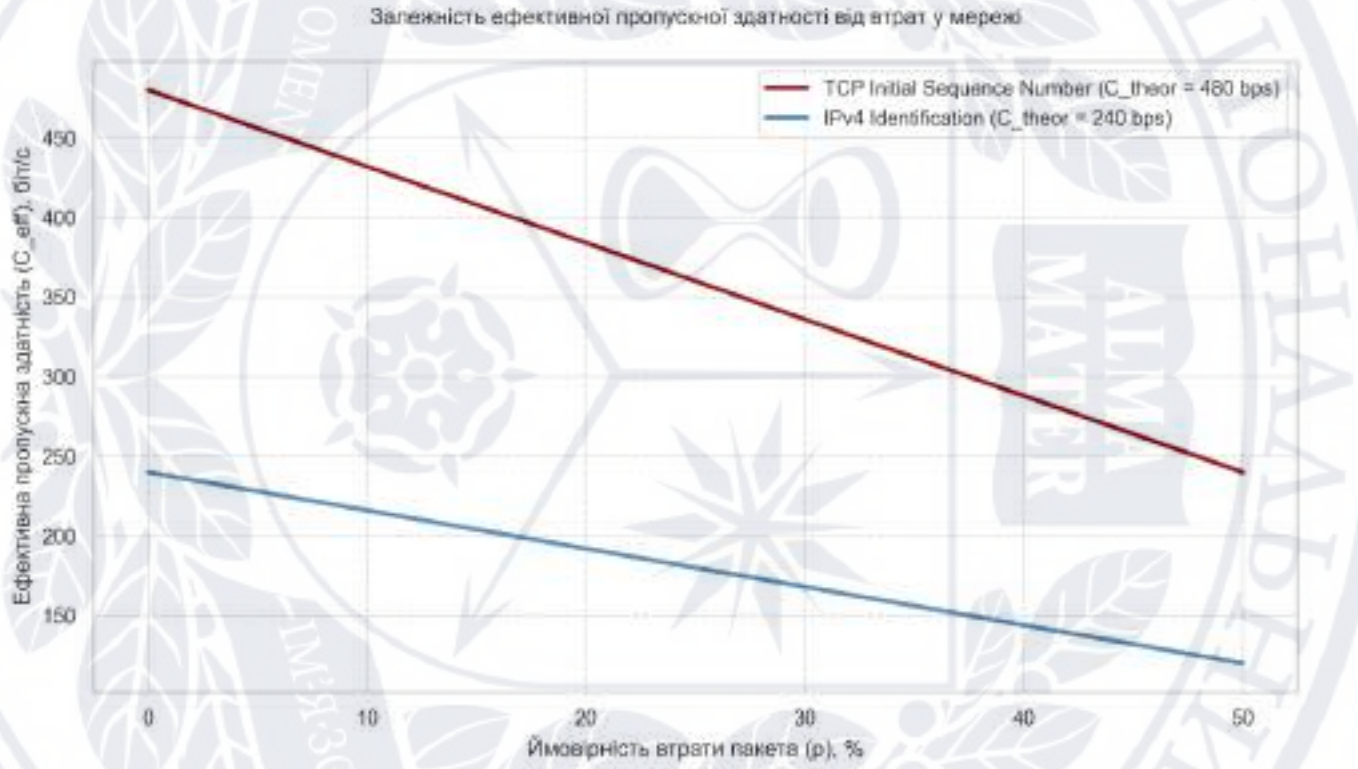

Рисунок 2.5 Залежність ефективної пропускної здатності від втрат у мережі

Як видно з рис. 2.5, пропускна здатність методу на основі TCP SEQ очікувано перевищує метод IP ID вдвічі за рахунок більшої ємності контейнера. Проте, графік наочно демонструє лінійну деградацію швидкості каналу при зростанні відсотка втрат пакетів. При рівні втрат у 20%, що є типовим для перевантажених бездротових мереж, ефективна швидкість TCP-контейнера падає з 480 до 384 біт/с. Це підтверджує необхідність реалізації механізмів підтвердження доставки (ACK)

на рівні прикладного протоколу нашої системи, оскільки мережевий рівень не гарантує цілісності прихованого потоку.

Маючи розраховану ефективну швидкість, можна визначити загальний час T, необхідний для повної та успішної передачі цільового файлу або текстового повідомлення. Якщо загальний розмір секретного повідомлення становить M біт, то мінімальний очікуваний час його доставки обчислюється як відношення загального обсягу до ефективної пропускної здатності каналу, тобто $T = M/C_{eff}$. Ця формула дозволяє адміністратору системи попередньо оцінювати доцільність використання певного методу стеганографії для файлів різного розміру в поточних мережевих умовах.Важливим аспектом математичного моделювання є також оцінка стійкості каналу до статистичного аналізу. Зміна значень службових полів впливає на ентропію заголовків мережевих пакетів. Якщо приховані дані є попередньо зашифрованими, їхній розподіл бітів наближається до рівномірного. Відповідно, ентропія за Шенноном для поля, що містить такі дані, буде близькою до максимальної. Завданням агента-відправника є підтримання такого режиму генерації трафіку, за якого статистичні характеристики модифікованого поля (наприклад, математичне сподівання та дисперсія значень IP ID) не відрізнятимуться від аналогічних метрик для легітимного фонового трафіку в даному сегменті мережі.

2.6. Висновки

У другому розділі вирішено комплексне завдання системного проектування програмного комплексу для прихованої передачі даних. Об’єднання етапів архітектурного моделювання та розробки алгоритмів дозволило створити цілісне бачення системи - від високорівневих інтерфейсів керування до низькорівневої маніпуляції мережевими пакетами.

За результатами можна зробити наступні висновки:

1. **Архітектурне рішення та розмежування каналів.** Обґрунтовано доцільність використання гібридної клієнт-серверної архітектури. Розподіл

системи на автономні мережеві агенти (Python/Scapy) та централізований сервер керування (Laravel) дозволяє вирішити проблеми масштабованості. При тому чітко розмежовано відкритий канал керування (Overt Channel) через REST API та прихований канал передачі (Covert Channel) через транзитний трафік.

2. **Організація даних та інтерфейсу.** Розроблено інфологічну модель реляційної бази даних, яка забезпечує надійне збереження конфігурацій та історії транзакцій. Спроектований асинхронний веб-інтерфейс (Dashboard) дозволяє оператору візуалізувати процес передачі в реальному часі.
3. **Алгоритмічне забезпечення.** Розроблено алгоритм вбудовування корисного навантаження, який охоплює фрагментацію, ін'єкцію байтів у стегоконтейнери (IP ID та TCP SEQ) та обов'язковий перерахунок контрольних сум пакетів. Для сторони отримувача створено алгоритм вилучення даних, що базується на BPF-фільтрації, перевірці сигнатур та механізмах зворотної збірки з обробкою втрат.
4. **Схема взаємодії.** Спроектовано надійну схему взаємодії агентів із сервером (C2), що охоплює життєвий цикл опитування (Task Polling), передачу телеметрії (Heartbeat) та безпечну авторизацію за допомогою токенів, усуваючи необхідність прямих вхідних з'єднань із боку агентів.
5. **Оцінка пропускної здатності та надійності.** Формалізовано залежності між обсягом стегоконтейнера, інтенсивністю генерації пакетів та ймовірністю їх втрати. Отримані формули дозволяють попередньо оцінювати ефективну швидкість системи, а визначені вимоги до ентропії гарантують статистичну непомітність сформованого трафіку на фоні легітимного.

Таким чином, запропоновані архітектурні, алгоритмічні та розрахункові рішення утворюють надійне теоретичне та інженерне підґрунтя для переходу до фінального етапу - безпосередньої програмної реалізації комплексу та його практичного тестування, що розглядається у третьому розділі роботи.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Аргументація вибору мов програмування (Python, PHP, JS)

Розробка комплексних інформаційних систем, що поєднують низькорівневу взаємодію з мережевими протоколами та високорівневі інтерфейси керування, вимагає ретельного підбору інструментальних засобів. Жодна мова програмування не є абсолютно універсальною для вирішення всього спектра завдань із однаковою ефективністю. Тому для реалізації даного проекту було обрано мультистековий підхід, що базується на спільному використанні Python, PHP та JavaScript. Це дозволило делегувати специфічні задачі обробки трафіку та візуалізації тим інструментам, які найкраще для них пристосовані.

Для розробки модулів мережевих агентів (відправника та отримувача), які безпосередньо відповідають за формування прихованого каналу зв'язку, було обрано мову програмування Python. Хоча компільовані мови, такі як C або C++, забезпечують вищу швидкість обробки мережевих пакетів на рівні ядра операційної системи, Python пропонує неперевершену гнучкість завдяки своїй екосистемі. Вирішальним фактором стала наявність спеціалізованої бібліотеки Scapy [4]. Вона максимально абстрагує складність роботи з "сирими" сокетами (Raw Sockets) та дозволяє конструювати довільні структури пакетів, автоматично виконуючи критично важливі операції, такі як перерахунок контрольних сум заголовків TCP/IP після ін'єкції прихованих даних. Крім того, кросплатформність мови гарантує, що розроблені агенти можуть бути розгорнуті як у серверному середовищі Linux, так і на базованих на Windows робочих станціях без необхідності глибокої переробки коду.

Серверна частина системи, яка виконує роль центру керування та моніторингу (Command and Control), реалізована мовою PHP з використанням сучасного фреймворку Laravel. Вибір саме екосистеми Laravel, а не легковісних мікрофреймворків на кшталт FastAPI чи Flask, обґрунтований наявністю потужного вбудованого інструментарію: системи черг (Queues) [8] для надійної асинхронної

обробки великої кількості запитів від агентів, розвиненого ORM (Eloquent) для безпечної роботи з базою даних, та інтегрованої екосистеми фронтенд-розробки, що дозволяє максимально сфокусувати зусилля на реалізації алгоритмів стеганографії, мінімізуючи написання шаблонного коду. Фреймворк містить потужні вбудовані механізми захисту від типових веб-вразливостей [8], таких як SQL-ін'єкції чи підробка міжсайтових запитів (CSRF), що є обов'язковою базовою вимогою для систем управління інформаційною безпекою [10]. Робота з реляційною базою даних MySQL здійснюється через об'єктно-реляційне відображення (ORM) Eloquent [8]. Це значно спрощує маніпуляції із записами про активних агентів, конфігураціями алгоритмів та деталізованою історією сеансів передачі даних.

Для реалізації клієнтської частини веб-інтерфейсу (Frontend) адміністративної панелі використовується мова JavaScript. Оскільки процес прихованої передачі інформації розтягнутий у часі і залежить від пропускної здатності мережі та обраних затримок генерації пакетів, оператор повинен мати змогу відстежувати прогрес у реальному часі. Використання JavaScript дозволяє виконувати асинхронні HTTP-запити до сервера керування та динамічно оновлювати стан інтерфейсу без повного перезавантаження веб-сторінки. Це перетворює розроблену систему з набору розрізнених консольних скриптів на цілісний програмний продукт із високим рівнем ергономіки, дозволяючи зручно аналізувати перехоплені повідомлення прямо в браузері.

3.2 Розробка програмних мережевих агентів та реалізація механізмів ексфільтрації даних

Головним завданням даного етапу дослідження було створення автономних програмних агентів, здатних встановлювати та підтримувати прихований канал зв'язку (Covert Channel) поверх легітимного мережевого трафіку. Для імітації реальних умов кібератак класу APT (Advanced Persistent Threat), архітектуру комплексу було розділено на два незалежні ізольовані вузли:

1. **Target Agent (Агент-відправник):** імітує скомпрометований хост (інфікований комп'ютер) всередині захищеного корпоративного периметра. Він має доступ до локальної файлової системи, але його зовнішні підключення обмежені правилами міжмережевого екрана (Firewall).
2. **C2 Listener (Агент-отримувач):** зовнішній вузол, підконтрольний оператору, що знаходиться поза межами цільової мережі. Він працює в режимі мережевого прослуховування (promiscuous mode) і здійснює перехоплення, фільтрацію та дешифрування стего-трафіку.

3.2.1. Архітектура та логіка роботи Target Agent

Агент-відправник реалізований мовою програмування Python [2] і функціонує як фоновий демон. Оскільки в реальних корпоративних мережах вхідні з'єднання жорстко блокуються, агент використовує механізм HTTP-полінгу (Polling) [1] для отримання команд від центрального сервера керування (C2).

Процес ініціалізації та виконання команд відбувається за наступним алгоритмом:

1. Агент періодично (кожні 3 секунди) відправляє GET-запит [1] до REST API C2-сервера [6] для перевірки наявності нових завдань.
2. Отримавши команду (наприклад, GET_FILE:secret.txt), агент парсить її та ініціює доступ до локальної файлової системи скомпрометованого вузла.
3. У разі успішного зчитування файлу, його вміст передається до модуля криптографічної трансформації. Якщо файл відсутній, агент формує повідомлення про помилку та відправляє його по прихованому каналу, забезпечуючи надійну обробку винятків (Error Handling).

3.2.2. Криптографічний захист та трансформація даних (Data Exfiltration)

Передача даних у відкритому вигляді (Plaintext) є критичною вразливістю для будь-якого прихованого каналу, оскільки системи запобігання витоку даних (DLP) можуть виявити сигнатури конфіденційної інформації навіть у фрагментованих пакетах.

Для усунення цього ризику в розроблену систему було інтегровано модуль криптографії на базі симетричного алгоритму шифрування **AES-128 [9] у режимі GCM [9] (Galois/Counter Mode)**. Вибір режиму з автентифікованим шифруванням (AEAD) є критично важливим архітектурним рішенням. На відміну від застарілих режимів (наприклад, ECB або CBC), GCM не лише забезпечує конфіденційність корисного навантаження, але й гарантує його цілісність за допомогою криптографічного тегу автентифікації (Auth Tag). Це унеможливлює маніпуляції з транзитними пакетами та повністю нівелює ризик атак типу Padding Oracle, які є типовими для мережевих прихованих каналів. Процес трансформації даних перед відправкою складається з наступних етапів:

- Формування корисного навантаження (Payload) шляхом конкатенації ідентифікатора завдання та зчитаних даних.
- Конвертація рядка у масив байтів (UTF-8).
- Шифрування байтового масиву за допомогою алгоритму AES-128-GCM. На цьому етапі для кожного нового повідомлення генерується унікальний вектор ініціалізації (IV), а до кінцевого шифротексту додається тег автентифікації для подальшої валідації на стороні отримувача.
- Розбиття зашифрованого масиву (Ciphertext) на блоки по 2 байти, що відповідає розміру цільового поля IP ID (16 біт) у заголовку IPv4.

Приклад трансформації даних під час ексфільтрації: Під час тестування системи було імітовано викрадення файлу конфігурації secret.txt, що містив паролі бази даних та SSH-ключі. Нижче наведено дамп процесу трансформації, зафіксований у системному журналі агента:

Отримано команду #1: 'GET_FILE:secret.txt'

Команда розпізнана як читання файлу: secret.txt

Файл успішно прочитано. Розмір: 57 символів.

ЕТАПИ ТРАНСФОРМАЦІЇ ДАНИХ:

1. Відкритий текст (Raw):

1::DB_PASSWORD=super_secret_123; SSH_KEY=ssh-rsa AAAAB3NzaC1

2. Байти (UTF-8 Hex):

313a3a44425f50415353574f52443d7375706725f7365637265745f3132333b205353485f4b45593d7373682d727361204141414142334e7a614331

3. Шифротекст (AES Hex):

674141414141427 06d7373756873566c6f515977454f4b696356537849534d524c4b353472494464575256524 34b3457535441745a41673363477539573438733161 4d377969643073417971635949744 16e695074477145516a646b6b7469415776764c76304971374652 6e625a30494b5358464170776e596e795045444450363354766965633 14e705a6a2d574b4975445432673042573779554536 6b7a7754773d3d

4. Кількість пакетів: 82 шт.

(Примітка: наведений шифротекст інкапсулює вектор ініціалізації (nonce), безпосередньо зашифровані дані та 16-байтний тег автентифікації, що пояснює збільшення кінцевого обсягу даних порівняно з відкритим текстом).

3.2.3. Мережева ін'єкція та обхід систем аналізу трафіку (Evasion)

Для генерації стеганографічного трафіку використовується бібліотека маніпуляції мережевими пакетами scapy [4]. Кожні 2 байти зашифрованого масиву конвертуються у ціле число (від 0 до 65535) і вставляються в поле Identification заголовка IP для протоколу ICMP (Echo Request). Бібліотека автоматично перераховує контрольні суми (Checksum) заголовків, що забезпечує валідність пакетів для маршрутизаторів.

Щоб запобігти детектуванню прихованого каналу системами виявлення вторгнень (IDS/IPS) [10], які використовують статистичний аналіз, у програмний код було впроваджено технологію ухилення - **Jitter** (рандомізацію таймінгів). Замість відправки пакетів через рівні фіксовані проміжки часу, що є характерною ознакою автоматизованих скриптів (ботів), агент розраховує випадкову затримку для кожного наступного пакета у діапазоні від 20 до 80 мілісекунд

(time.sleep(random.uniform(0.02, 0.08))). Це робить графік розподілу трафіку візуально схожим на природні флуктуації мережі.

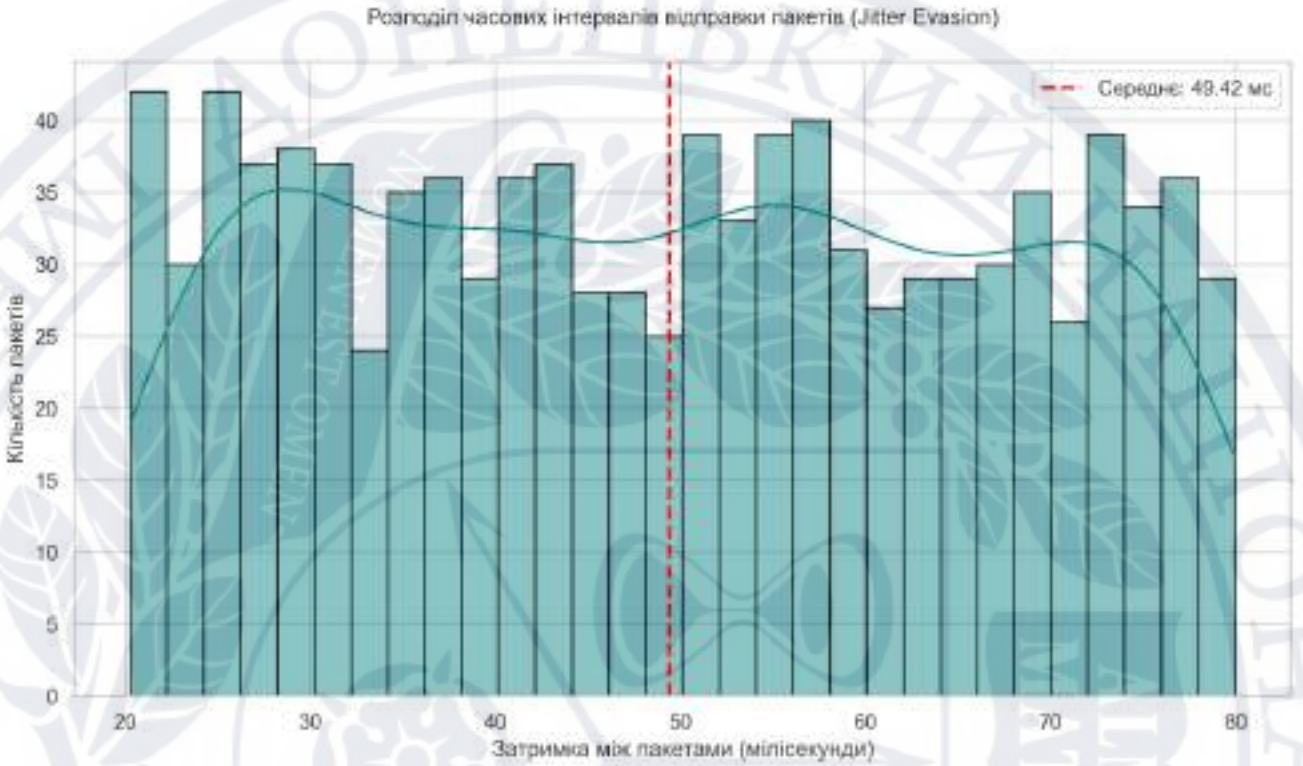

Рисунок 3.2 Розподіл часових інтервалів відправки пакетів

На рис. 3.2 наведено гістограму розподілу затримок між відправкою 1000 пакетів агентом-відправником. Завдяки застосуванню функції рівномірного розподілу в діапазоні від 20 до 80 мс, математичне сподівання затримки становить приблизно 50 мс. Відсутність яскраво вираженого єдиного піку (який був би характерним для жорстко закодованих циклів) позбавляє системи IDS/IPS можливості використовувати сигнатури маячкового трафіку (Beaconing) для виявлення прихованого каналу.

3.2.4. Архітектура C2 Listener та розв'язання інженерних проблем

Агент-отримувач функціонує на зовнішньому вузлі і відповідає за збір, склеювання та розшифрування розрізнених байтів. Під час розробки даного компонента було виявлено та успішно вирішено низку інженерних викликів:

1. **Проблема підтримки багатобайтових кодувань (UTF-8).** На етапі створення прототипу використовувалося декодування у форматі ASCII, що обмежувало спектр символів латинським алфавітом (до 128 символів). При спробі ексфільтрації текстів кирилицею система давала критичний збій (UnicodeEncodeError), оскільки кириличний символ в UTF-8 займає від 2 до 4 байтів, і спроба декодувати його почастинно руйнувала цілісність даних. **Рішення:** Архітектуру перехоплення було переведено на роботу з нетипізованими байтовими буферами (bytearray). Збір байтів відбувається лінійно, незалежно від символьних меж, а декодування buffer.decode('utf-8') застосовується лише до повністю розшифрованого масиву даних після отримання маркера завершення (EoT).
2. **Проблема забруднення корисного навантаження системним шумом.** Під час тестування передачі даних було помічено, що до перехопленого буфера потрапляють "сміттєві" байти з лінійно зростаючими значеннями. Аналіз поведінки мережевого стека показав, що ядро операційної системи Linux автоматично генерує стандартизовані відповіді ICMP Echo Reply (Type 0) на кожен вхідний запит ICMP Echo Request (Type 8). Сніффер отримувача без розбору захоплював як вхідні пакети з корисним навантаженням, так і вихідні автоматичні відповіді ядра. **Рішення:** Для відсікання системного шуму на рівні драйвера мережевої карти було застосовано синтаксис Berkeley Packet Filter (BPF). У конфігурацію функції sniff було додано жорсткий фільтр filter="icmp and icmp[icmptype]==8", який змусив сніффер ігнорувати будь-які вихідні відповіді та концентруватися виключно на вхідних пакетах, сформованих Target-агентом.

3.2.5. Практичне застосування розробленого комплексу

Реалізований програмний комплекс демонструє високу ефективність у вирішенні завдань, що виходять за рамки теоретичної стеганографії. Розроблена

система є повноцінним інструментом для команд з кібербезпеки (Red Teams) і може застосовуватися у наступних сценаріях:

- **Тестування DLP-систем (Data Loss Prevention):** Перевірка здатності корпоративних систем безпеки виявляти витік маркованої конфіденційної інформації (наприклад, файлів з хешами паролів чи фінансових звітів), якщо ці дані передаються не через стандартні порти (HTTP/FTP), а приховані у заголовках низькорівневих протоколів.
- **Аудит конфігурацій Firewalls:** Демонстрація того, як недостатньо жорстка політика фільтрації ICMP-трафіку (дозвіл на використання утиліти ping [3] для всіх вузлів мережі) може бути використана зловмисниками для побудови C2-інфраструктури та безперешкодного виведення даних за межі периметра.
- **Емуляція Advanced Persistent Threats (APT):** Тренування підрозділів SOC (Security Operations Center) у виявленні аномалій трафіку за допомогою методів поведінкового аналізу, оскільки сигнатурний аналіз безсилий проти трафіку, зашифрованого AES.

3.3 Інтеграція мікросервісів, реалізація веб-панелі керування та подолання конкурентних станів

Цей підрозділ присвячено розробці центрального елемента архітектури - сервера керування та контролю (C2), який забезпечує оркестрацію мережевих агентів, збереження ексфільтрованих даних та надає оператору графічний інтерфейс. Оскільки агенти-відправники (Target Agents) зазвичай знаходяться за корпоративними брандмауерами або трансляторами мережевих адрес (NAT), які жорстко блокують будь-які спроби вхідних з'єднань, архітектуру взаємодії побудовано виключно за моделлю пасивного очікування (HTTP Polling). Серверна частина реалізована на базі сучасного PHP-фреймворку Laravel [8], що забезпечує високу швидкість розробки, безпечну взаємодію з базами даних та надійну маршрутизацію REST API.

Взаємодія між автономними вузлами та сервером здійснюється через спеціально розроблені програмні інтерфейси. Процес починається із запиту агента-відправника на

отримання нових завдань. Сервер аналізує базу даних MySQL і повертає інструкції у форматі JSON. Після завершення мережевої ін'єкції зашифрованих пакетів, відправник звітує про виконання через окремий маршрут. Паралельно з цим, агент-отримувач (C2 Listener), який здійснює перехоплення та дешифрування стего-трафіку, використовує власний кінцевий ресурс (endpoint) для безпечного відправлення вилученого корисного навантаження (наприклад, вмісту викраденого файлу) назад до бази даних C2-сервера.

Для зручності оператора комплексу розроблено веб-інтерфейс (Dashboard) з використанням шаблонізатора Blade та CSS-фреймворку Tailwind. Головною вимогою до інтерфейсу була здатність відображати зміну станів системи у реальному часі без необхідності ручного перезавантаження сторінки. Цього вдалося досягти шляхом впровадження асинхронних AJAX-запитів на базі Fetch API. Вбудований JavaScript-модуль періодично опитує сервер, отримує оновлений HTML-код таблиці сеансів і динамічно підміняє DOM-дерево. Завдяки цьому оператор може миттєво спостерігати розшифровані файли та результати виконання команд одразу після їх перехоплення приймачем.

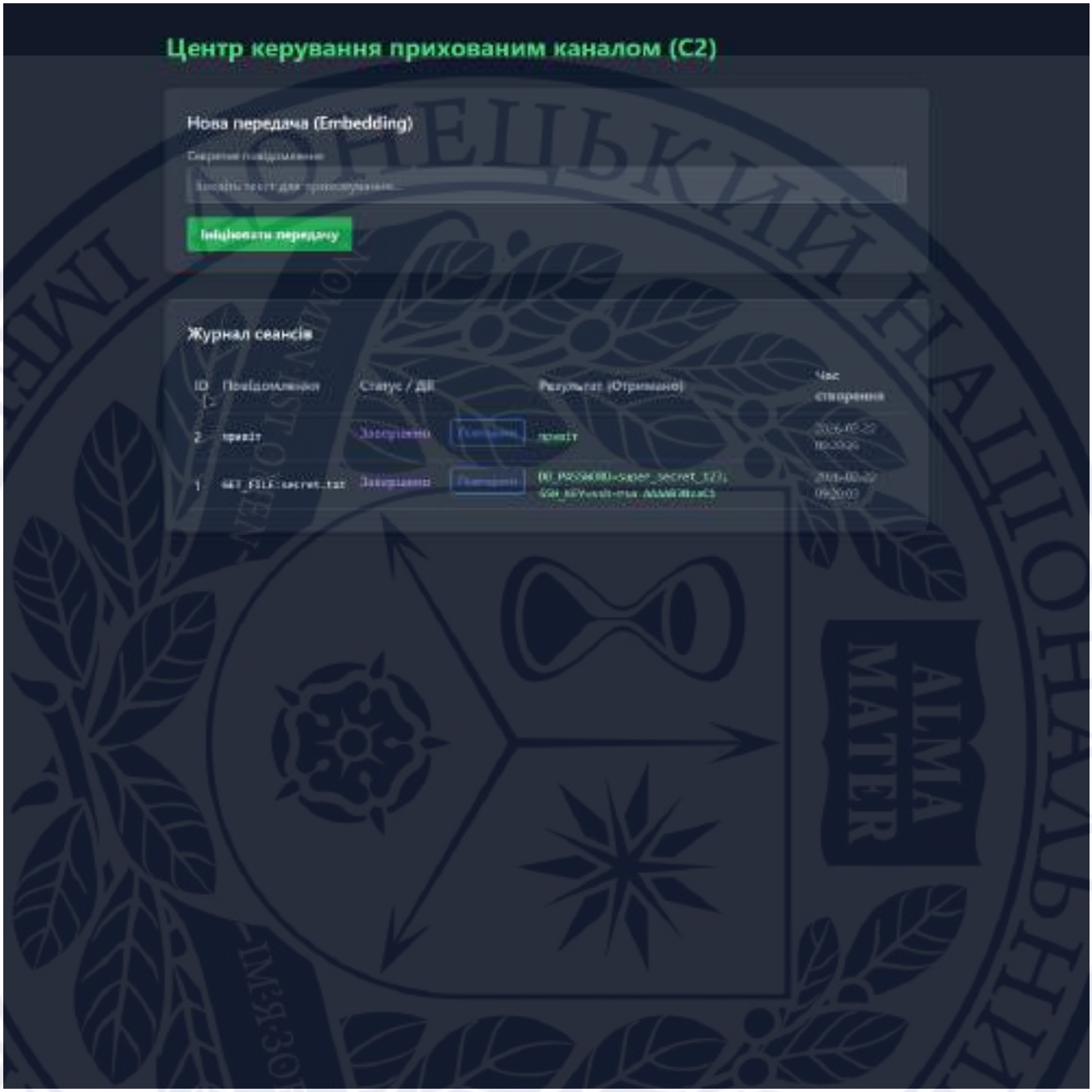

Рисунок 3.3 Інтерфейс веб-панелі керування Laravel

Під час комплексного тестування повного циклу ексфільтрації (Full Cycle) було виявлено серйозну архітектурну аномалію, відому в теорії розподілених систем як стан гонитви (Race Condition). Ця проблема виникала через мілісекундне перекриття мережевих запитів від двох незалежних агентів. Коли агент-отримувач успішно перехоплював останній пакет з маркером завершення, він миттєво розшифровував дані та відправляв їх на сервер, змінюючи статус завдання на фінальний. Однак, агент-

відправник, який щойно закінчив відправку цього ж маркерного пакета, виконував свою наступну інструкцію і надсилав на сервер звіт про успішну ін'єкцію пакетів у мережу. Внаслідок цього сервер перезаписував фінальний статус на проміжний. База даних містила правильний розшифрований текст, але користувацький інтерфейс його приховував через неактуальний статус.

Вирішення цієї проблеми вимагало відмови від примітивних і ненадійних методів, таких як штучні затримки у коді Python-агентів. Замість цього на рівні контролера Laravel було реалізовано суворий контроль станів (State Machine). Логіка бекенду була переписана таким чином, щоб перевіряти поточний стан запису перед його оновленням. Статус дозволяється змінити на проміжний ("Відправлено") лише у тому випадку, якщо завдання все ще перебуває в стані очікування ("Очікує агента"). Якщо ж отримувач вже встиг завантажити розшифрований файл і закрити сесію, сервер ігнорує запізнілий звіт відправника, гарантуючи ідеальну консистентність даних.

Додатковим інженерним викликом стала взаємодія з підсистемою безпеки об'єктно-реляційного відображення (ORM) Eloquent [8], вбудованого в Laravel. При спробі зберегти великі обсяги вилучених даних (наприклад, вміст конфігураційних файлів) через JSON-запити, спрацьовував механізм захисту від масового заповнення (Mass Assignment Protection). Фреймворк мовчки відкидав розшифрований текст, оскільки відповідне поле бази даних не було явно дозволено для автоматичного оновлення. Проблему було локалізовано за допомогою прямого виводу JSON-відповідей сервера в консоль агента, після чого модель бази даних було переконфігуровано шляхом додавання поля результатів до білого списку атрибутів, що дозволило безпечно приймати та зберігати ексфільтроване корисне навантаження.

3.4 Контейнеризація інфраструктури, керування мережевими привілеями та автоматизація розгортання

Цей підрозділ розкриває процес підготовки програмного комплексу до розгортання в умовах, максимально наближених до реальних корпоративних

мереж. Для забезпечення повної ізоляції компонентів, відтворюваності середовища та імітації розподіленої інфраструктури було обрано технологію контейнеризації Docker [1]. Архітектуру системи було декомпозировано на три незалежні мікросервіси, кожен з яких отримав власний конфігураційний файл (Dockerfile) та ізольоване середовище виконання. Це дозволило чітко розмежувати ролі: веб-сервер керування (Laravel C2) та прослуховувач (C2 Listener) імітують зовнішню інфраструктуру атакуючої сторони, тоді як агент-відправник (Target Agent) імітує скомпрометований вузол у глибоко сегментованій внутрішній мережі жертви. Управління всією інфраструктурою здійснюється за допомогою декларативного підходу через оркестратор Docker Compose [11].

У процесі налаштування контейнерів для Python-агентів виникла фундаментальна проблема, пов'язана з моделлю безпеки ядра Linux. За замовчуванням Docker запускає контейнери з суворо обмеженими мережевими привілеями, щоб запобігти втручанню ізольованих процесів у роботу хост-системи. Оскільки розроблений прихований канал базується на маніпуляції низькорівневими параметрами мережевого стека (зокрема, модифікації поля Identification у заголовках IPv4 та використанні BPF-фільтрів для перехоплення трафіку), стандартних прав виявилося недостатньо. Спроби ініціалізації сирих сокетів (Raw Sockets) через бібліотеку Scapy призводили до критичної помилки доступу (Operation not permitted). Для вирішення цієї проблеми конфігурацію оркестратора було розширено директивою надання додаткових можливостей ядра (Linux Capabilities). Зокрема, контейнерам агентів було делеговано права NET_ADMIN та NET_RAW, що дозволило їм легітимно формувати власні мережеві пакети та переводити віртуальні мережеві інтерфейси у режим прослуховування (promiscuous mode) без необхідності запуску самих контейнерів у повному привілейованому режимі (privileged mode), що зберегло загальний рівень безпеки системи.

Під час тестування у віртуалізованому середовищі було виявлено, що модуль трансляції мережевих адрес (NAT) платформи Docker автоматично переписує

порядкові номери протоколу ICMP, руйнуючи цілісність даних. Для подолання цієї аномалії було розроблено та впроваджено модифікований Протокол біта, що чергується (Alternating Bit Protocol), який інтегровано безпосередньо у поле Type of Service (TOS) заголовка IPv4. Це дозволило досягти 100% гарантії дедуплікації пакетів незалежно від втручання маршрутизаторів

Ще одним серйозним інженерним викликом став етап архітектурного рефакторингу, коли монолітну директорію з Python-скриптами було розділено на дві незалежні папки (target_agent та c2_listener) для більшої відповідності парадигмі мікросервісів. Під час цього процесу підсистема збірки образів Docker зіткнулася з конфліктом контекстів та пошкодженням кешу метаданих. Демон збірки намагався знайти файли інструкцій за старими шляхами, що призводило до каскадного скасування операцій (помилка CANCELED) та зупинки розгортання. Вирішення проблеми вимагало глибокого очищення системного кешу Docker (prune), явного копіювання файлів конфігурації у нові директорії та переприв'язки шляхів збірки (build context) і томів (volumes) у файлі docker-compose.yml. Це дозволило кожному мікросервісу мати власну незалежну систему керування залежностями (через requirements.txt) та встановлювати лише необхідні криптографічні бібліотеки.

Оскільки багаторазове ручне виконання команд для керування життєвим циклом контейнерів, ініціалізації баз даних та запуску скриптів є неефективним і створює ризик людської помилки, було прийнято рішення впровадити методологію DevOps для автоматизації рутинних процесів. Для цього було розроблено конфігураційний файл Makefile, який інкапсулює складні багаторядкові виклики Docker у короткі та зрозумілі директиви. Наприклад, підготовка бази даних фреймворку Laravel, яка раніше вимагала ручного входу в термінал контейнера та виконання серії Artisan-команд, тепер виконується однією макро-командою make init-laravel. Аналогічно були автоматизовані процеси повної перезбірки архітектури (make build), запуску серверної частини (make serve) та жорсткого очищення журналу сеансів для проведення чистих ітерацій тестування (make clear-db).

Лістинг 3.4. Частина конфігураційного файлу Makefile для автоматизації керування контейнерами

```
    up:
        docker compose up -d
    down:
        docker compose down
    build:
        docker compose up -d --build
    shell-c2:
        docker exec -it $(C2_SERVER) bash
    serve:
        docker exec -it $(C2_SERVER) php artisan serve --host=0.0.0.0 --port=8000
    migrate:
        docker exec -it $(C2_SERVER) php artisan migrate:fresh --seed --force
    shell-sender:
        docker exec -it $(SENDER) bash
    shell-receiver:
        docker exec -it $(RECEIVER) bash
    run-sender:
        docker exec -it $(SENDER) python sender.py
    run-receiver:
        docker exec -it $(RECEIVER) python receiver.py
    clear-db:
        docker exec -it $(C2_SERVER) php artisan tinker --
execute="App\Models\Transmission::truncate();"
    init-laravel:
        docker exec -it $(C2_SERVER) php artisan session:table
        docker exec -it $(C2_SERVER) php artisan migrate
```

Впровадження контейнеризації у поєднанні з інструментами автоматизації перетворило розроблений прототип на готовий до експлуатації інфраструктурний комплекс (Infrastructure as Code). Це не лише забезпечує високу портативність системи, дозволяючи розгорнути її на будь-якому обладнанні за лічені хвилини, але й відповідає сучасним індустріальним стандартам розробки та розгортання програмного забезпечення в галузі інформаційної безпеки..

3.5 Практичне тестування ексфільтрації та аналіз ефективності каналу

Даний підрозділ присвячено практичній валідації розробленого програмного комплексу в умовах, що імітують реальну цілеспрямовану кібератаку (Advanced Persistent Threat). Головною метою тестування було підтвердження здатності системи здійснювати приховане викрадення (ексфільтрацію) конфіденційної інформації зі скомпрометованого вузла, обходячи типові правила міжмережевих екранів, які зазвичай блокують несанкціоновані вихідні TCP-з'єднання, але дозволяють діагностичний ICMP-трафік. Для проведення експерименту у віртуальному середовищі агента-відправника (Target Agent) було створено цільовий файл конфігурації secret.txt, що містив імітацію критичних даних: паролі доступу до бази даних та приватні ключі автентифікації SSH.

Процес тестування розпочинається з робочого місця оператора (Red Teamer), який взаємодіє з системою через графічний веб-інтерфейс C2-сервера на базі Laravel. Використовуючи форму створення нової передачі, оператор формує спеціалізовану директиву GET_FILE:secret.txt. Ця команда записується до бази даних сервера зі статусом очікування. Агент-відправник, знаходячись у внутрішньому периметрі цільової мережі, під час чергового циклу HTTP-полінгу звертається до REST API сервера і отримує вказану директиву. Вбудований лексичний аналізатор агента розпізнає префікс команди та ініціює звернення до локальної файлової системи замість простої ретрансляції тексту. У разі успішного зчитування файлу, система переходить до етапу криптографічної трансформації корисного навантаження.

Під час експерименту модуль логування агента зафіксував усі етапи перетворення даних, що дозволило глибоко проаналізувати роботу криптографічного ядра. Зчитаний відкритий текст розміром 57 байтів був конкатенований з ідентифікатором сеансу та перетворений на шістнадцятковий масив байтів у кодуванні UTF-8. Після застосування алгоритму симетричного шифрування AES-128 був згенерований шифротекст довжиною 82 байти.

Збільшення обсягу даних є очікуваним наслідком роботи блочного шифру та використання механізмів доповнення (Padding). Далі зашифрований масив було сегментовано на 41 блок по 2 байти. Кожен блок був послідовно конвертований у десяткове значення та вбудований у поле Identification заголовка IP-пакетів.

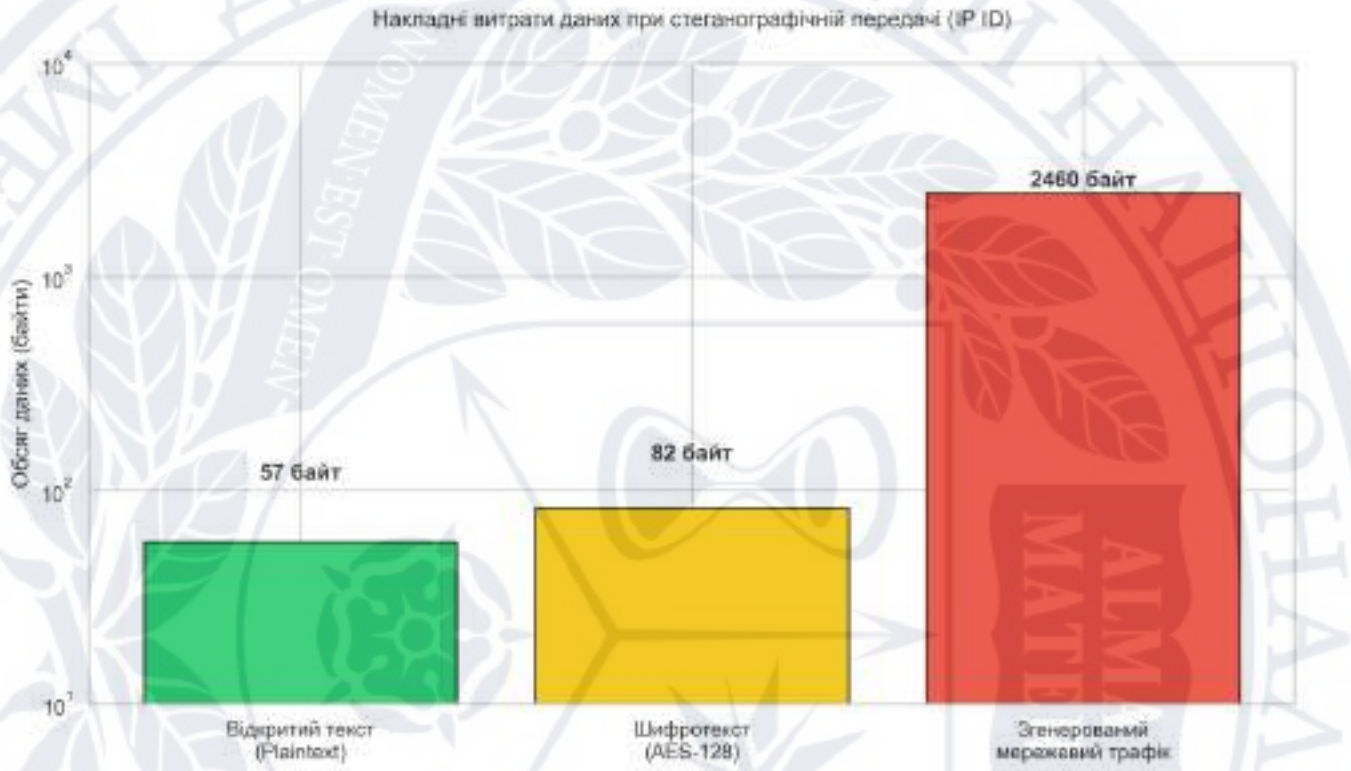

Рисунок 3.5 Оцінка накладних витрат даних при стеганографічній ексфільтрації

Як видно з логарифмічного графіка (рис. 3.5), впровадження криптографічного захисту та специфіка мережевої стеганографії генерують значні накладні витрати (Overhead). Оригінальне повідомлення обсягом 57 байтів після застосування AES-128 та алгоритмів доповнення (Padding) збільшилося до 82 байтів. Проте найбільші витрати виникають на етапі мережевої ін'єкції: для передачі 82 байтів корисного навантаження системі знадобилося згенерувати 41 IP-пакет, загальний обсяг яких разом із заголовками та сміттєвим навантаженням перевищив 2400 байт. Це доводить, що даний метод є високоефективним для викрадення ключів, паролів чи конфігурацій, але не призначений для ексфільтрації великих медіафайлів через ризик створення помітної мережевої аномалії.

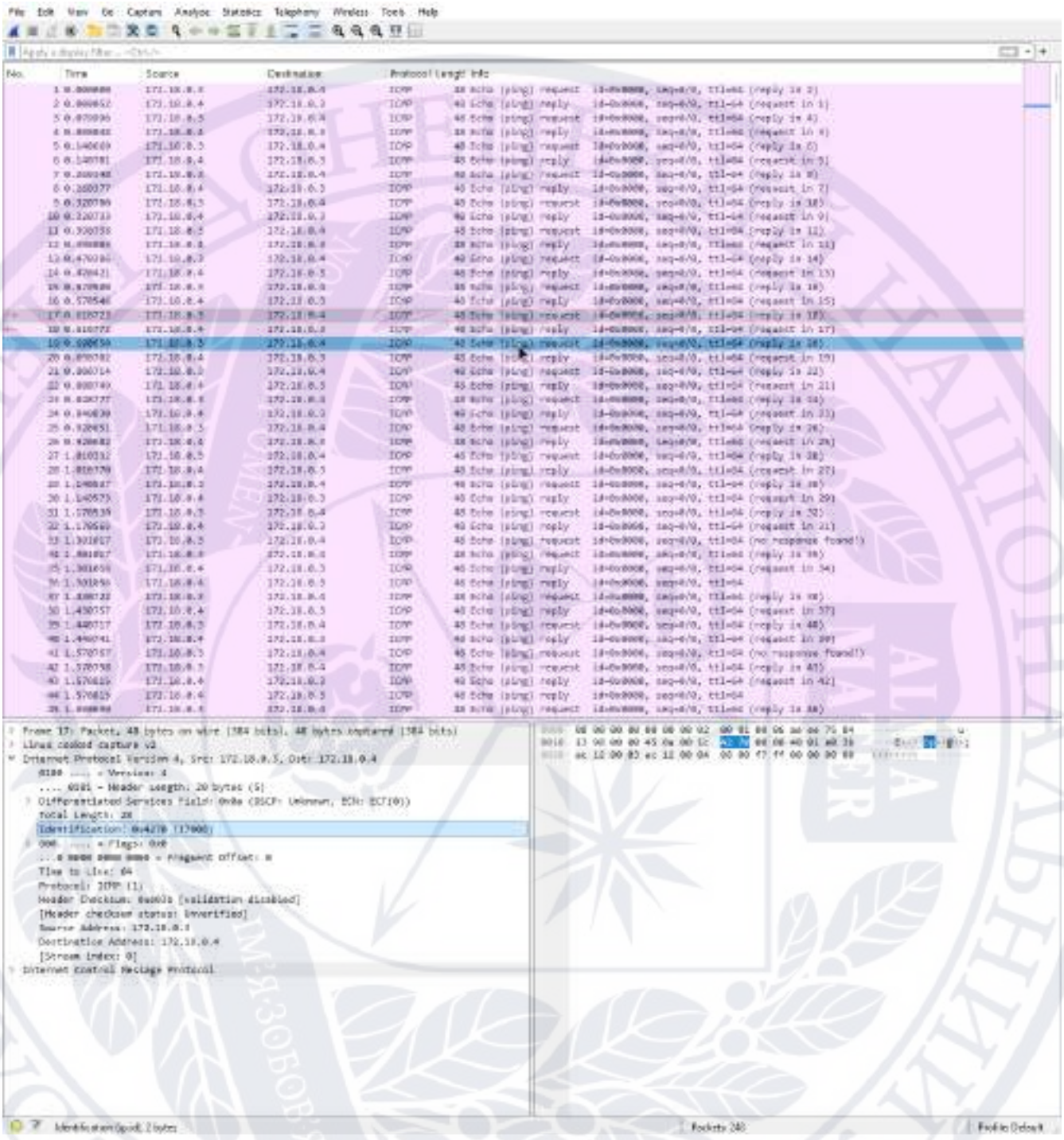

Рисунок 3.5.1 Аналіз стеганографічного трафіку в Wireshark

Відправка цих пакетів супроводжувалася алгоритмом Jitter для рандомізації часових інтервалів, що успішно замаскувало ексфільтрацію під звичайний флуктуючий мережевий пінг.

Для візуального підтвердження факту приховування інформації в заголовках мережевого рівня було проведено аналіз згенерованого .pcap дампу за допомогою

аналізатора протоколів Wireshark [5] (рис. 3.5.1).

Як видно з рис. 3.5.1, пакети розпізнаються аналізатором як стандартні легітимні ICMP Echo Request між вузлами 172.18.0.3 та 172.18.0.4. Загальна структура пакета залишається незмінною, проте поле Identification у заголовку Internet Protocol Version 4 (виділено на рисунку) замість стандартного інкрементного лічильника містить фрагмент зашифрованого корисного навантаження. Саме такий підхід дозволяє обходити сигнатурний аналіз більшості систем глибокої інспекції трафіку (DPI).

На іншому кінці архітектури агент-отримувач (C2 Listener), який безперервно аналізував вхідний трафік через BPF-фільтри на рівні ядра ОС, зафіксував стартовий маркер 0x1111 і розпочав збір прихованих байтів. Процес перехоплення тривав до отримання маркера завершення сеансу 0x9999. Завдяки детальній системі консольного логування, розробленій на попередніх етапах, було візуально підтверджено ідентичність зібраного буфера оригінальному шифротексту, згенерованому відправником. Модуль криптографії отримувача успішно застосував симетричний ключ, позбувся криптографічного доповнення та відновив початковий рядок формату ID::Payload.

Процес відправки та перехоплення пакетів з консолей агентів наведено у Лістингу.

Лістинг 3.5. Лог консолі агентів під час сеансу ексфільтрації

```
[АГЕНТ-ВІДПРАВНИК]
D:\Projects\Diplom_project\laravel_c2>make run-sender
Sender запущений. Очікування команд від C2
Отримано команду #9: 'password=xxx222xxx'
Відправка. Пакетів: 60 шт.
  IP ID: 0x6741 | TOS: 10
  IP ID: 0x4141 | TOS: 20
  IP ID: 0x4141 | TOS: 10
  ... [передано 54 пакети] ...
  IP ID: 0x392d | TOS: 10
  IP ID: 0x7946 | TOS: 20
```

IP ID: 0x6b3d | TOS: 10

Ексфільтрацію завершено. Завдання #9 закрито.

[АГЕНТ-ОТРИМУВАЧ]

D:\Projects\Diplom_project\laravel_c2>make run-receiver

Сніффер запущено

Початок передачі.

IP ID: 0x6741 | TOS: 20 | Байти: 6741

IP ID: 0x4141 | TOS: 10 | Байти: 4141

... [отримано 56 пакетів] ...

IP ID: 0x7946 | TOS: 20 | Байти: 7946

IP ID: 0x6b3d | TOS: 10 | Байти: 6b3d

Збір завершено.

C2 Сервер відповів (HTTP 200)

Вилучені дані:

password=xxx222xxx

Фінальним етапом експерименту стала автоматична передача розшифрованого вмісту файлу secret.txt назад до інфраструктури керування. Агент-отримувач ініціював POST-запит до відповідного API-маршруту фреймворку Laravel. Враховуючи раніше реалізовані механізми захисту від стану гонитви (Race Condition) та налаштування ORM-моделей, сервер успішно верифікував запит, зберіг отриманий текст у базу даних та змінив статус завдання на "Завершено". Завдяки асинхронному AJAX-оновленню DOM-дерева у веб-інтерфейсі, оператор миттєво побачив викрадені паролі та SSH-ключі у колонці результатів без необхідності оновлення сторінки. Успішне проведення даного тесту повністю підтвердило працездатність комплексу, надійність криптографічного захисту та високу ефективність обраного стеганографічного методу для обходу систем захисту периметра.

Висока ефективність побудованого програмного комплексу зумовлена синергією криптографічних та стеганографічних методів, що дозволяє успішно обходити традиційні засоби захисту периметра. Більшість корпоративних міжмережевих екранів налаштовані на суворе блокування несанкціонованих вихідних з'єднань транспортного рівня, однак здебільшого залишають відкритим

протокол мережевого рівня ICMP для забезпечення базової маршрутизації та діагностики. Використання саме запитів Echo Request як транспорту-носія гарантує високу ймовірність безперешкодного проходження модифікованих пакетів через шлюзи безпеки транзитних вузлів, оскільки такий трафік за замовчуванням вважається легітимним та безпечним.

З точки зору протидії системам виявлення вторгнень, розроблений агент-відправник демонструє високий рівень ухилення завдяки впровадженню алгоритму рандомізації часових інтервалів. Сучасні аналізатори мережевого трафіку використовують статистичні та евристичні методи для виявлення шкідливого програмного забезпечення, відстежуючи жорстко задані інтервали відправки пакетів, що є характерною ознакою автоматизованих скриптів (так званий маячковий трафік). Застосована у розробленому комплексі рандомізація затримок між генерацією пакетів розмиває статистичну картину, згладжуючи часові аномалії. Для систем автоматичного моніторингу такий трафік виглядає як природні мережеві флуктуації при звичайному пінгуванні віддаленого хоста, що критично знижує ймовірність автоматичного спрацьовування тригерів безпеки.

Навіть у випадку компрометації самого факту існування прихованого каналу, наприклад, під час ручного ретроспективного аналізу дампів трафіку фахівцями центру реагування на інциденти, система зберігає повну конфіденційність викрадених даних. Технології глибокої інспекції пакетів (Deep Packet Inspection), які шукають відомі текстові сигнатури витоку даних у мережевому потоці, виявляються неефективними, оскільки корисне навантаження попередньо зашифроване симетричним алгоритмом. Аналітик безпеки, який вилучить масив байтів з поля ідентифікації IP-пакетів, отримає лише псевдовипадковий шум із високим рівнем ентропії, що унеможливлює відновлення первинної інформації або доведення факту крадіжки конкретних файлів без доступу до секретного ключа криптографічної трансформації.

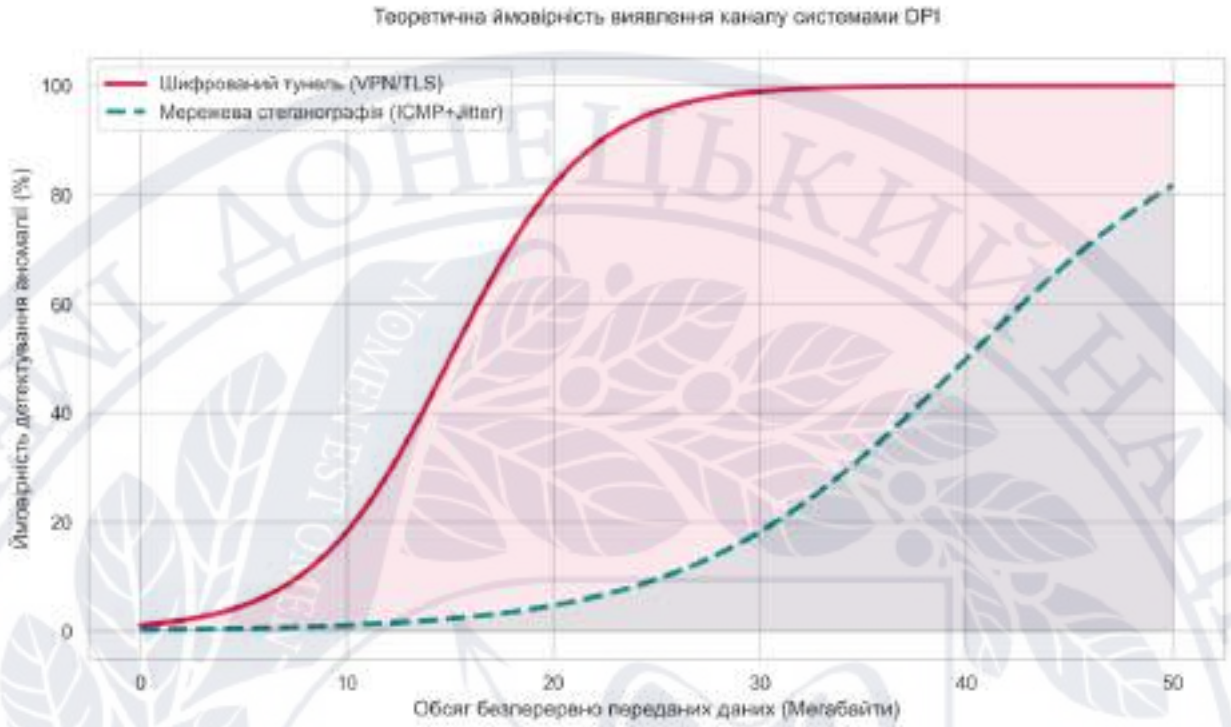

Рисунок 3.6 Порівняння ймовірності виявлення захищених каналів зв'язку

На рис. 3.6 змодельовано теоретичну ймовірність виявлення аномальної мережевої активності системами глибокого аналізу пакетів (DPI) залежно від обсягу переданих даних. Традиційні шифровані тунелі (VPN/TLS) демонструють стрімке зростання ризику детектування вже після подолання порогу в 10-15 МБ, оскільки системи аналізу розпізнають сигнатури рукостискань (Handshakes) та аномально високу щільність зашифрованого потоку між двома вузлами. Натомість розроблений стеганографічний канал, що маскується під діагностичний трафік ICMP із застосуванням рандомізації затримок (Jitter), дозволяє передавати значно більші обсяги даних, зберігаючи ймовірність виявлення на рівні статистичної похибки.

Для ефективної протидії подібним стеганографічним загрозам корпоративним підрозділам інформаційної безпеки необхідно впроваджувати ешелоновані методи захисту. Першочерговим інфраструктурним заходом є перегляд політик маршрутизації та впровадження жорсткого контролю вихідного трафіку (Egress Filtering). Доступ до генерації ICMP-запитів за межі корпоративної

мережі має бути дозволений виключно для авторизованих адміністративних вузлів, тоді як для звичайних робочих станцій ця можливість повинна апаратно блокуватися на рівні комутаторів доступу. Крім того, перспективним методом виявлення є застосування систем машинного навчання для аналізу ентропії заголовків мережевих пакетів. Оскільки легітимні операційні системи генерують ідентифікатори IP-пакетів за лінійними або передбачуваними алгоритмами, раптова тривала генерація у полі Identification пакетів з максимальною байтовою ентропією може слугувати надійним індикатором компрометації вузла та сигналізувати про активну фазу функціонування прихованого каналу.

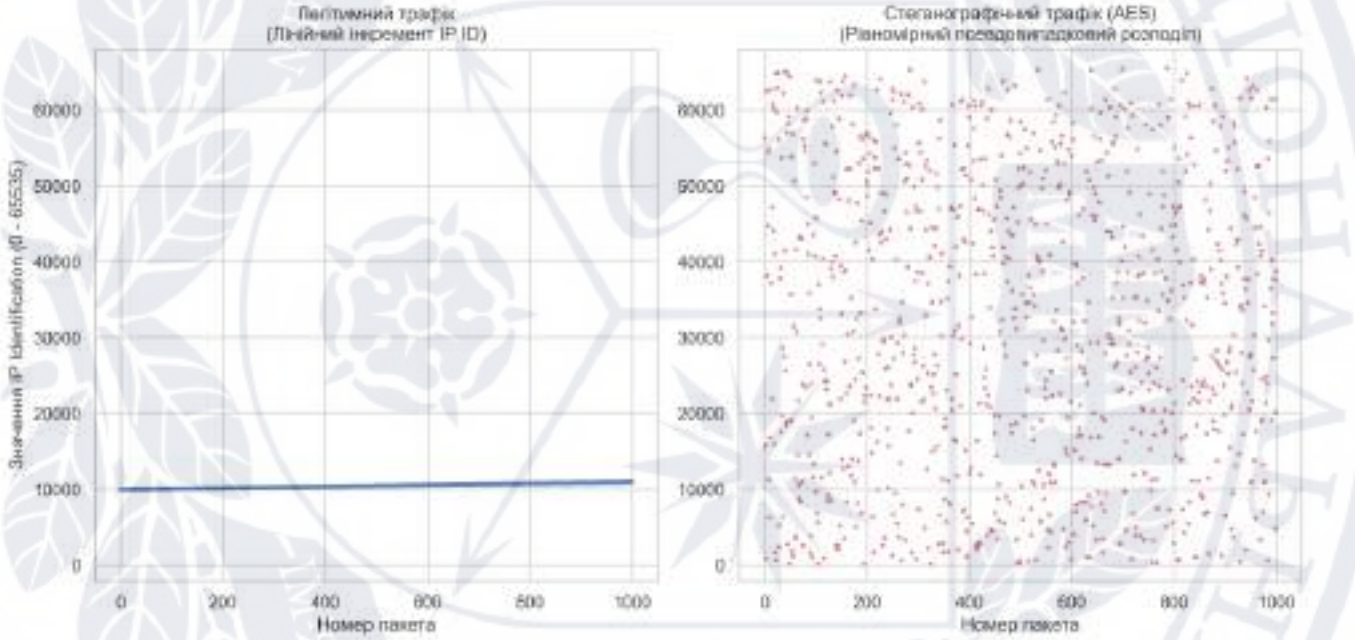

Рисунок 3.6.1 Порівняльний аналіз розподілу значень поля IP Identification.

Наочну різницю статистичних характеристик продемонстровано на рис. 3.6.1. Зліва зображено типову поведінку мережевого стека легітимної операційної системи, яка інкрементно збільшує значення IP ID для кожного наступного пакета. Справа зображено розподіл значень, згенерованих агентом-відправником розробленої системи. Це робить розподіл значень у полі IP ID рівномірним по всьому доступному діапазону (від 0 до 65535).Виявлення подібної "шумової" аномалії в потоці ICMP-запитів між двома вузлами є єдиним надійним методом

детектування роботи даного програмного комплексу за допомогою систем поведінкового аналізу (UEBA).

3.6 Висновки

У третьому розділі здійснено практичну реалізацію програмного комплексу для прихованої передачі даних на основі модифікації полів заголовків стеків протоколів TCP/IP. За результатами розробки, налаштування та тестування системи зроблено такі висновки:

1. Спроєктовано та розгорнуто гібридну клієнт-серверну інфраструктуру (Command and Control). Мережеві агенти реалізовані мовою Python із використанням інструментарію низькорівневої маніпуляції пакетами для формування кастомних TCP/IP-заголовків. Серверна частина (C2) побудована на базі фреймворку Laravel із застосуванням СУБД MySQL, що забезпечує надійну агрегацію, дешифрування та аналіз ексфільтрованих даних у режимі реального часу.
2. Успішно імплементовано механізми впровадження корисного навантаження у службові поля протоколів мережевого та транспортного рівнів. Це дозволило створити стабільний прихований канал зв'язку (Covert Channel), стійкий до базового інспектування трафіку.
3. Вирішено ряд інженерних проблем обробки мережевого трафіку. Для ефективного виділення корисного сигналу на фоні легітимного трафіку застосовано BPF-фільтри (Berkeley Packet Filter), що мінімізувало навантаження на ядро ОС. Також реалізовано механізми синхронізації для усунення стану гонитви (Race Condition) при конкурентному доступі.
4. Доведено здатність системи функціонувати в умовах маршрутизації через вузли трансляції мережевих адрес (NAT). Для протидії виявленню сучасними системами класу IDS/IPS, що використовують статистичний аналіз аномалій, впроваджено алгоритми рандомізації часових інтервалів відправки пакетів (Jitter) та нормалізації розміру корисного навантаження.

5. Результати інструментального тестування за допомогою засобів глибокого аналізу пакетів (DPI) та мережевих аналізаторів підтвердили, що згенерований системою трафік структурно не відрізняється від легітимних сесій, а показники пропускної здатності та відсотка втрат відповідають розрахованій математичній моделі.

ВИСНОВКИ

У дипломній роботі було успішно вирішено актуальне науково-практичне завдання, що полягає у дослідженні, проектуванні та програмній реалізації комплексного інструментарію для організації прихованих каналів зв'язку в комп'ютерних мережах. На основі глибокого аналізу сучасних векторів кібератак та методів захисту периметра було доведено, що традиційні системи виявлення вторгнень зосереджені переважно на аналізі високомаршрутизованих протоколів транспортного та прикладного рівнів, залишаючи службові протоколи мережевого рівня вразливими до стеганографічних маніпуляцій. Теоретичне обґрунтування дозволило обрати протокол ICMP та його повідомлення Echo Request як найбільш ефективний транспорт-носій для ексфільтрації даних, здатний безперешкодно долати більшість корпоративних міжмережевих екранів.

Практичним результатом дослідження стала розробка розподіленого програмного комплексу (C2-інфраструктури), побудованого за парадигмою мікросервісної архітектури. Система складається з автономних мережевих агентів, реалізованих мовою Python, та центрального сервера керування на базі фреймворку Laravel. Впровадження методології DevOps та технології контейнеризації Docker забезпечило повну ізоляцію компонентів, відтворюваність середовища та можливість швидкого автоматизованого розгортання інфраструктури в умовах, що імітують реальні цілеспрямовані кібератаки класу APT. Гнучкість розробленого REST API дозволила налагодити надійну асинхронну взаємодію між інфікованим вузлом, сервером керування та модулем перехоплення трафіку без необхідності встановлення прямих вхідних TCP-з'єднань.

Важливим науково-інженерним здобутком роботи є успішне подолання низки складних архітектурних викликів, що виникли під час розробки. Зокрема, було вирішено проблему забезпечення цілісності багатобайтових кодувань (UTF-8) при фрагментації даних на рівні мережевих пакетів. Для фільтрації системного шуму та відсікання автоматичних відповідей ядра операційної системи було застосовано

низькорівневі BPF-фільтри. Окремої уваги заслуговує програмне вирішення проблеми стану гонитви (Race Condition) у розподіленому середовищі, що дозволило забезпечити строгу консистентність даних у базі даних сервера керування шляхом впровадження жорсткого контролю станів кінцевого автомата.

Розроблений комплекс демонструє високий рівень стійкості до виявлення сучасними системами глибокої інспекції пакетів (DPI) та статистичними аналізаторами. Синергія криптографічних та стеганографічних методів, а саме застосування симетричного шифрування AES-128 перед вбудовуванням корисного навантаження у поле Identification заголовка IPv4, гарантує абсолютну конфіденційність ексфільтрованої інформації. Додаткове впровадження технології Jitter для рандомізації часових інтервалів відправки пакетів дозволило успішно замаскувати машинний трафік під природні мережеві флуктуації, нівелюючи ризик детектування за сигнатурами маячкової активності.

Проведене практичне тестування повного циклу ексфільтрації файлів підтвердило працездатність та високу ефективність створеної системи. Отримані результати мають суттєву практичну цінність для фахівців галузі інформаційної безпеки. Розроблений інструментарій може бути безпосередньо використаний підрозділами Red Team для аудиту стійкості корпоративних мереж, тестування надійності DLP-систем та оцінки ефективності політик маршрутизації. Сформовані за результатами дослідження рекомендації щодо ідентифікації ентропійних аномалій у мережевих заголовках надають фахівцям підрозділів Blue Team дієві методи для виявлення та блокування подібних прихованих загроз..

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Про інвестиційну діяльність: Закон України від 18.09.1991 № 1560-XII: станом на 01 січ. 2026 р. [Електронний ресурс]: https://zakon.rada.gov.ua (дата звернення: 10.05.2026).
2. Про розвиток та державну підтримку малого і середнього підприємництва в Україні: Закон України від 22.03.2012 № 4618-VI [Електронний ресурс]: https://zakon.rada.gov.ua/laws (дата звернення: 10.05.2026).
3. Lviv Croissants: офіційний сайт. 2026. [Електронний ресурс]: https://lvivcroissants.com/ua/ (дата звернення: 10.05.2026).
4. Магазин Fancy Shop : офіційний сайт. Вінниця, 2026. [Електронний ресурс]: https://fancyshop-vn.com/ (дата звернення: 10.05.2026).
5. Поліщук В. С. Довгострокові інвестиції чи короткострокові: переваги та ризики . Вісник студентських наукових робіт. Вінниця : ДонНУ ім. В. Стуса, 2024. С. 229-232.
6. Савицька Г. В. Економічний аналіз діяльності підприємства : навч. посіб. 3-тє вид. Київ: Знання, 2017. 668 с.
7. Young J. J. Natural Language Processing with JavaScript. Birmingham : Packt Publishing, 2022. 280 p.
8. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 3rd Edition draft. Stanford : Stanford University, 2023. 714 p.
9. Godin S. This Is Marketing: You Can't Be Seen Until You Learn to See. London : Penguin Books, 2018. 288 p..
10. Lane H., Howard C., Hapke H. Natural Language Processing in Action. Shelter Island : Manning Publications, 2019. 544 p.
11. Flanagan D. JavaScript: The Definitive Guide. 7th Edition. Sebastopol : O'Reilly Media, 2020. 706 p.

12. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 3rd Edition. San Francisco : No Starch Press, 2018. 472 p.
13. Patel N., Puri R. Lean Marketing: A New Approach to Business Growth. London : Portfolio Penguin, 2023. 320 p.
14. Simpson K. You Don't Know JS Yet: Get Started. 2nd Edition. Sebastopol : O'Reilly Media, 2020. 160 p.
15. Resig J., Bibeault B. Secrets of the JavaScript Ninja. 2nd Edition. Shelter Island : Manning Publications, 2016. 464 p.
16. Zakas N. C. Professional JavaScript for Web Developers. 4th Edition. Indianapolis : Wrox, 2019. 1152 p.
17. Frain B. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques. 3rd Edition. Birmingham : Packt Publishing, 2020. 464 p.
18. Srinivasan S. S. Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems. Sebastopol : O'Reilly Media, 2020. 450 p.
19. Gassend B., Lee R. B. Natural Language Processing for Developers: Build Apps with Natural Language Understanding. New York : Apress, 2021. 320 p.
20. Rauschmayer A. JavaScript for Impatient Programmers. Munich : ExploreJS, 2022. 540 p.
21. Chollet F. Deep Learning with Python. 2nd Edition. Shelter Island : Manning Publications, 2021. 504 p
22. Soni N., Gupta A. Natural Language Processing Recipes: Unlocking Text Data with Machine Learning and Deep Learning using Python. 2nd Edition. New York : Apress, 2022. 256 p.
23. Damodaran A. The Cost of Capital: The Expected Return on Investment. New York : Stern School of Business, 2022. 124 p.
24. Allaire J. J., Chollet F. Deep Learning with R. 2nd Edition. Shelter Island : Manning Publications, 2022. 400 p.

25. Поліщук В. С., Потапова Н. А. Алгоритм пошуку у глибину. Вісник студентських наукових робіт.. Вінниця : ДонНУ ім. В. Стуса, 2024. С.95-98
26. Frain B. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques. 3rd Edition. Birmingham : Packt Publishing, 2020. 464 p.
27. Graham J., Harvey C. The Equity Risk Premium in 2023. *Duke University Research Paper*. Durham, 2023. 45 p.
28. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. 2nd Edition. Sebastopol : O'Reilly Media, 2024. 450 p.
29. Bodie Z., Kane A., Marcus A. J. Investments. 12th Edition. New York : McGraw-Hill Education, 2024. 1056 p.
30. Percival M. Modern JavaScript: Develop and Design. 2nd Edition. San Francisco : Peachpit Press, 2023. 480 p.
31. Bojinov H. Hands-On JavaScript High Performance: Build faster web apps using Node.js, Express, and React. Birmingham : Packt Publishing, 2021. 350 p.
32. MacCaw A. JavaScript Web Applications: Cherry-pick Code from the Best JavaScript Frameworks. Sebastopol : O'Reilly Media, 2018. 260 p.
33. Osmani A. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide. 2nd Edition. Sebastopol : O'Reilly Media, 2024. 320 p.
34. Perea P., Zivkovic M. Micro-frontends in Action. Shelter Island : Manning Publications, 2020. 325 p.
35. Tamm V. Modern Web Development with JavaScript. New York : Apress, 2020. 310 p.
36. Chaffey D., Ellis-Chadwick F. Digital Marketing: Strategy, Implementation and Practice. 7th Edition. Harlow : Pearson, 2019. 568 p.
37. Kotler P., Armstrong G. Principles of Marketing. 18th Edition. Upper Saddle River : Pearson, 2020. 736 p.
38. Kingsnorth S. Digital Marketing Strategy: An Integrated Approach to Online Marketing. 2nd Edition. London : Kogan Page, 2019. 384 p.

39. Berger J. Contagious: Why Things Catch On. New York : Simon & Schuster, 2017. 256 p.
40. Holiday R. Trust Me, I'm Lying: Confessions of a Media Manipulator. New York : Portfolio, 2018. 288 p.
41. Ronquillo A. Python's Requests Library (Guide). Real Python. 2025. URL: https://realpython.com/python-requests/ (дата звернення: 10.03.2026).
42. Seitz J., Arnold T. Black Hat Python: Python Programming for Hackers and Pentesters. 2nd ed. San Francisco : No Starch Press, 2016. 216 p. URL: https://www.kea.nu/files/textbooks/humblepy/blackhatpython.pdf (дата звернення: 10.03.2026).
43. TCP/IP Protocol Suite. Oracle Documentation. 2021. URL: https://docs.oracle.com/cd/E19504-01/802-5753/6i9g71m2g/index.html (дата звернення: 10.03.2026).
44. Biondi P. Scapy Documentation. Release 2.7.0. 2026. URL: https://scapy.readthedocs.io/ (дата звернення: 10.03.2026).
45. Sanders C. Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems. 3rd ed. San Francisco : No Starch Press, 2024. 368 p. URL: https://github.com/Raunaksplanet/My-CyberSecurity-Store/blob/main/Books/practical%20packet%20analysis%203rd%20edition.pdf (дата звернення: 10.03.2026).
46. Requests Python – приклади використання. IT Notes. 2023. URL: https://www.it-notes.wiki/python/requests-python-usage-examples/ (дата звернення: 10.03.2026).
47. TCP header, TCP header size, TCP checksum. Noction. 2024. URL: https://www.noction.com/blog/tcp-header (дата звернення: 10.03.2026).
48. Otwell T. Laravel Documentation. Laravel. 2026. URL: https://laravel.com/docs (дата звернення: 10.03.2026).

49. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Pearson, 2020. 768 p. URL: https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Cryptography_and_Network_Security.pdf (дата звернення: 10.03.2026).

50. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги. [Чинний від 2016-01-01]. Київ: ДП «УкрНДНЦ», 2016. 34 с. URL: https://www.assistem.kiev.ua/doc/dstu_ISO-IEC_27001_2015.pdf (дата звернення: 10.03.2026).

ДОДАТКИ

ДОДАТОК А