НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях

До захисту допущено:
ВО завідувача кафедри
_____ Валерій ПРАВИЛО
«_____»_____2026 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційно-комунікаційні технології»

спеціальності 172 Телекомунікації та радіотехніка

на тему: Метод автоматизованого аналізу мережевого трафіку в реальному часі

Виконав: здобувач вищої освіти 4 курсу, групи ТІ-21
Осадчий Богдан Володимирович _____

Науковий керівник: доцент кафедри ІТТ НН ІТС, кандидат технічних наук,
доцент Суліма Світлана Валеріївна _____

Рецензент: доцент кафедри ТК НН ІТС, кандидат технічних наук
доцент Міночкін Д.А _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____

Київ – 2026 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях

Рівень вищої освіти – перший (бакалаврський)

Освітньо-професійна програма «Інформаційно-комунікаційні технології»

спеціальності 172 Телекомунікації та радіотехніка

ЗАТВЕДЖУЮ

ВО завідувача кафедри

_____ Валерій ПРАВИЛО

«____»_____2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Осадчому Богдану Володимировичу

1. Тема дипломної роботи: Метод автоматизованого аналізу мережевого трафіку в реальному часі, науковий керівник роботи доцент кафедри ІТТ НН ІТС, кандидат технічних наук, доцент Суліма Світлана Валеріївна -
затверджені наказом по університету від 26.05.2026 р. № 1912-с

2. Строк подання студентом дипломної роботи 06.06.2026 р.

3. Об’єкт дослідження: процес моніторингу, перехоплення та класифікації мережевого трафіку на рівнях L2–L7 моделі OSI в інформаційно-телекомунікаційних системах.

4. Вихідні дані до роботи: гібридні методи детекції мережевих аномалій, евристичні патерни класифікації кіберзагроз, алгоритми обчислення кругової затримки (RTT) та пропускної здатності в реальному часі.

5. Перелік завдань, які потрібно розробити:

1. Проаналізувати існуючі програмні засоби моніторингу мережевого трафіку та визначити їхні функціональні обмеження.

2. Обґрунтувати гібридний метод детекції аномалій на основі поєднання статистичного аналізу ковзного вікна та евристичних правил.
3. Розробити алгоритми пасивного вимірювання показників якості обслуговування (QoS) мережі без генерації тестового трафіку.
4. Спроектувати модульну архітектуру конвеєру обробки даних програмного комплексу.
5. Реалізувати програмні модулі пасивного захоплення пакетів та детекції п'яти базових класів мережевих загроз.
6. Розробити веб-інтерфейс користувача для візуалізації та моніторингу мережевих подій у реальному часі.
7. Провести експериментальне тестування розробленого рішення, оцінити точність детекції та часові показники швидкодії системи.

6. Перелік ілюстративного матеріалу, (із зазначенням плакатів, презентацій тощо): Презентація -18 слайдів, 27 таблиць, 19 рисунків

7. Дата видачі завдання: 06.09.2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Узгодження організаційних питань з науковим керівником	24.09.25-25.09.25	Виконано
2	Узгодження теми роботи та початок роботи над першим розділом	04.02.26-04.03.26	Виконано
3	Робота над першим, другим та третім розділами	20.04.26-10.05.26	Виконано
4	Робота над четвертим розділом. Планування розробки власного рішення	11.05.24-17.05.26	Виконано
5	Розробка власного рішення та його перевірка	15.03.26 - 20.05.26	Виконано
6	Висновки про ефективність рішення	21.05.26 - 31.05.26	Виконано
7	Оформлення дипломної роботи	06.11.26 - 12.06.26	Виконано

8	Підготовка презентації до захисту	01.06.26 - 12.06.26	Виконано
9	Захист дипломної роботи	16.06.26 - 19.06.26	Виконано

Здобувач вищої освіти _____ Богдан ОСАДЧИЙ

Науковий керівник роботи _____ Світлана СУЛІМА

РЕФЕРАТ

Дипломна робота містить 95 сторінок тексту, 19 рисунків, 27 таблиць та 35 літературних джерел.

Актуальність теми:

Зростання кількості кіберзагроз у корпоративних мережах і відсутність доступних інструментів автоматизованого моніторингу для малого і середнього бізнесу обумовлюють необхідність розробки легковагової системи виявлення аномалій у реальному часі. Існуючі рішення або потребують ручного аналізу трафіку або вимагають розгортання дорогої серверної інфраструктури.

Метою роботи є підвищення ефективності та оперативності виявлення мережевих кіберзагроз і оцінки якості обслуговування (QoS) шляхом розробки гібридного методу та легковагового програмного комплексу автоматизованого аналізу трафіку, що працює в реальному часі без залучення виділених серверних потужностей.

Об'єктом дослідження є процес аналізу мережевого трафіку в корпоративних мережах.

Предметом дослідження є методи автоматизованого виявлення аномалій у мережевому трафіку на основі статистичного та евристичного аналізу пакетів.

Методами дослідження є статистичний аналіз трафіку з використанням ковзного вікна часу, евристичні алгоритми класифікації аномалій, інформаційно-ентропійний аналіз DNS-запитів за формулою Шеннона та експериментальне тестування у контрольованому середовищі.

Отримані результати. Розроблено програмний комплекс TrafficAnalyzer що реалізує гібридний метод детекції п'яти класів мережевих загроз: DDoS, PortScan, BruteForce, ARP Spoofing і DNS Tunneling. Система розгортається на стандартному персональному комп'ютері без додаткової інфраструктури і забезпечує час реакції від 2 до 30 секунд від початку атаки до сповіщення у веб-інтерфейсі.

Ключові слова: МЕРЕЖЕВИЙ ТРАФІК, ВИЯВЛЕННЯ АНОМАЛІЙ, КІБЕРЗАГРОЗИ, МОНІТОРИНГ МЕРЕЖІ, DDOS, PORTSCAN, ARP SPOOFING, DNS TUNNELING, BRUTEFORCE, SCAPY, FLASK.

ABSTRACT

The bachelor`s thesis contains 95 pages of text, 19 figures, 27 tables and a list of 35 references.

Relevance of the topic:

The growing number of cyberthreats in corporate networks and the lack of accessible automated monitoring tools for small and medium-sized businesses create a need for a lightweight real-time anomaly detection system. Existing solutions either require manual traffic analysis or demand expensive dedicated server infrastructure.

The objective of the work is to improve the efficiency and promptness of detecting network cyberthreats and evaluating Quality of Service (QoS) by developing a hybrid method and a lightweight software system for automated traffic analysis that operates in real time without involving dedicated server infrastructure.

The object of the study is the process of network traffic analysis in corporate networks.

The subject of the study is the methods of automated anomaly detection in network traffic based on statistical and heuristic packet analysis.

The research methods include statistical analysis using a sliding time window, heuristic anomaly classification algorithms, information-entropy analysis of DNS queries based on the Shannon formula and experimental testing in a controlled network environment.

Results obtained. A software system TrafficAnalyzer was developed that implements a hybrid detection method for five classes of network threats: DDoS, PortScan, BruteForce, ARP Spoofing and DNS Tunneling. The system is deployed on a standard personal computer without additional infrastructure and provides a response time of 2 to 30 seconds from the start of an attack to an alert in the web interface.

Keywords: NETWORK TRAFFIC, ANOMALY DETECTION, CYBERTHREATS, NETWORK MONITORING, DDOS, PORTSCAN, ARP SPOOFING, DNS TUNNELING, BRUTEFORCE, SCAPY, FLASK.

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ACK	Acknowledgment (прапорець TCP, що підтверджує отримання даних)
API	Application Programming Interface (інтерфейс програмування застосунків)
ARP	Address Resolution Protocol (протокол визначення MAC-адреси за IP)
BPF	Berkeley Packet Filter (фільтр пакетів на рівні ядра ОС)
DDoS	Distributed Denial of Service (розподілена атака на відмову в обслуговуванні)
DNS	Domain Name System (система доменних імен)
HTTP	HyperText Transfer Protocol (протокол передачі гіпертексту)
HTTPS	HTTP Secure (захищена версія HTTP з TLS-шифруванням)
IP	Internet Protocol (протокол мережевого рівня)
MAC	Media Access Control (фізична адреса мережевого пристрою)
QoS	Quality of Service (якість обслуговування мережі)
RDP	Remote Desktop Protocol (протокол віддаленого робочого столу)
RST	Reset (прапорець TCP, що скидає з'єднання)
RTT	Round Trip Time (час передачі пакета туди і назад)
SIEM	Security Information and Event Management (система управління інформаційною безпекою)
SMB	Server Message Block (протокол доступу до мережевих ресурсів)
SSH	Secure Shell (протокол захищеного віддаленого доступу)
SYN	Synchronize (прапорець TCP, що ініціює з'єднання)
TCP	Transmission Control Protocol (протокол транспортного рівня з гарантією доставки)
UDP	User Datagram Protocol (протокол транспортного рівня без гарантії доставки)

ВСТУП

Сучасні корпоративні мережі є постійною ціллю кіберзагроз різних класів: від розподілених атак на відмову в обслуговуванні до прихованих каналів передачі даних через DNS. За даними галузевих досліджень кількість зафіксованих мережевих атак щорічно зростає і малий та середній бізнес не має можливості захиститись за допомогою дорогих комерційних рішень.

Існуючі інструменти моніторингу вирішують задачу лише частково. Пасивні сніфери такі як Wireshark потребують постійної присутності адміністратора і ручного аналізу трафіку.[1] Системи інфраструктурного моніторингу Nagios і Zabbix контролюють доступність обладнання але не аналізують вміст пакетів.[2,3] Комерційні SIEM-платформи забезпечують комплексний захист але потребують виділеної серверної інфраструктури і річних ліцензійних витрат від кількох тисяч доларів що робить їх недоступними для більшості підприємств.[4, 5]

Метою роботи є підвищення ефективності та оперативності виявлення мережевих кіберзагроз і оцінки якості обслуговування (QoS) шляхом розробки гібридного методу та легковагового програмного комплексу автоматизованого аналізу трафіку, що працює в реальному часі без залучення виділених серверних потужностей.

Об'єктом дослідження є процес аналізу мережевого трафіку в корпоративних мережах.

Предметом дослідження є методи автоматизованого виявлення аномалій у мережевому трафіку на основі статистичного та евристичного аналізу пакетів.

Для досягнення мети вирішено такі завдання: проведено аналіз існуючих рішень і виявлено незаповнену нішу, розроблено математичний апарат гібридного методу детекції, реалізовано програмний комплекс з п'ятьма детекторами і веб-інтерфейсом, проведено тестування у контрольованому середовищі і виконано порівняльний аналіз з існуючими рішеннями.

Практичне значення роботи полягає у тому що розроблений комплекс може бути розгорнутий у корпоративній мережі будь-якого масштабу без додаткових

витрат на інфраструктуру і ліцензування і забезпечує автоматичне сповіщення адміністратора про виявлені загрози менше ніж за 30 секунд від початку атаки.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ІНСТРУМЕНТІВ МОНІТОРИНГУ МЕРЕЖЕВОГО ТРАФІКУ

1.1. Огляд інструментів пасивного моніторингу (Wireshark, tcpdump)

Для вирішення базових задач оперативної діагностики, локалізації джерел несправностей, а також первинного виявлення аномальної активності в мережах традиційно застосовують інструменти пасивного аналізу. Вони функціонують за принципом перехоплення та декодування сирих пакетів і належать до класу пакетних сніферів. Основними представниками цього програмного забезпечення є консольна утиліта tcpdump та програмний комплекс із графічною оболонкою Wireshark. [1]

Архітектурний принцип роботи обох інструментів є однаковим і базується на використанні системних бібліотек низькорівневого захоплення пакетів. Для Unix-подібних операційних систем застосовується бібліотека libpcap, тоді як у сімействі Windows використовується її адаптована версія під назвою Npcap. Механізм пасивного моніторингу передбачає обов'язкове переведення мережевого адаптера у режим нерозбірливого захоплення, який у технічній літературі називають promiscuous mode.

У штатному режимі роботи мережева карта виконує апаратну фільтрацію кадрів на канальному рівні моделі OSI. Вона аналізує фізичну MAC-адресу призначення у заголовку кожного отриманого кадру та порівнює її з власною MAC-адресою мережевого адаптера. Якщо ці адреси не збігаються, мережева карта автоматично відкидає такий кадр і не передає його далі в операційну систему. Винятком є лише широкомовні кадри та пакети групової розсилки.

Активація режиму promiscuous mode повністю вимикає цей вбудований фільтр. Мережевий адаптер починає копіювати абсолютно всі зафіксовані у фізичному середовищі кадри, незалежно від того, якому пристрою в мережі вони адресовані. Після цього весь масив даних через бібліотеку захоплення

направляється безпосередньо в простір користувача для аналізу прикладним програмним забезпеченням.

Консольна утиліта tcpdump є класичним легковаговим інструментом, який часто інтегрують безпосередньо в операційні системи маршрутизаторів та комутаторів. Головна перевага tcpdump полягає у відсутності графічної оболонки, що забезпечує мінімальне споживання оперативної пам'яті та ресурсів процесора. Це дозволяє запускати утиліту для тривалого моніторингу на критично завантажених вузлах мережі без ризику порушити стабільність їхньої роботи.

Для відсікання надлишкового інформаційного шуму в tcpdump вбудовано механізм низькорівневої фільтрації BPF (Berkeley Packet Filter).[6] Ці фільтри компілюються у вигляді байт-коду і виконуються прямо всередині ядра операційної системи. Завдяки цьому пакети, які не відповідають заданим критеріям, відкидаються одразу в момент перехоплення. Такий підхід економить ресурси комп'ютера, оскільки системі не доводиться копіювати непотрібні байти з простору ядра в робочу пам'ять програми.

Проте утиліта tcpdump має суттєві обмеження, які заважають використовувати її як повноцінний автономний засіб детекції мережевих аномалій у реальному часі. Виведення результатів аналізу здійснюється у вигляді неперервного текстового потоку в консоль адміністратора, де відображаються лише базові поля заголовків пакетів. Така форма представлення інформації є незручною для швидкого виявлення динамічних мережевих атак, наприклад розподіленого флуду (DDoS) або швидкісного сканування портів (PortScan).

Адміністратор змушений або візуально відслідковувати аномалії в лавиноподібному потоці консольних рядків, або записувати весь трафік у файли дампів формату pcap для подальшої ретроспективної обробки. Це повністю виключає можливість автоматичного реагування на кіберзагрози в межах однієї секунди.

Програмний комплекс Wireshark є більш функціональним рішенням, призначеним для глибокого аналізу вмісту пакетів за технологією DPI (Deep Packet Inspection). [1] Його архітектура дозволяє детально розбирати структуру протоколів на всіх рівнях моделі OSI. Wireshark підтримує функції об'єднання розрізнених пакетів у повноцінні логічні сесії, що дає змогу вивчати хронологію взаємодії між клієнтом та сервером. Також комплекс містить статистичний рушій, здатний будувати графіки інтенсивності потоків та виводити загальні матриці мережевих з'єднань.

Незважаючи на високу аналітичну цінність, використання Wireshark як постійного автоматизованого центру безпеки в реальних мережах є технічно неефективним з двох основних причин:

1. Висока ресурсомісткість процесу аналізу. Для забезпечення швидкого пошуку, сортування та фільтрації даних через графічне вікно Wireshark змушений утримувати копію кожного перехопленого пакета разом із результатами його аналізу безпосередньо в оперативній пам'яті. В умовах реального мережевого навантаження, коли в системі активно працюють користувачі, передаються файли або функціонує IP-телефонія, Wireshark здатний повністю вичерпати доступний об'єм ОЗУ за короткий проміжок часу. Це призводить до аварійного завершення роботи програми або до зависання операційної системи моніторингової станції.
2. Відсутність інтегрованих механізмів автоматичної автономної детекції атак. Wireshark за своєю концепцією є інструментом пасивного відображення та ручної діагностики. У ньому немає вбудованих статистичних модулів або евристичних аналізаторів, які могли б у автоматичному режимі самостійно ідентифікувати ознаки атаки, такі як підміна адрес при ARP-спуфінгу чи перебір паролів методом брутфорсу. Програма не вміє самостійно генерувати тривожні сповіщення. Виявлення загрози повністю залежить від людського фактора, оскільки адміністратор має сам аналізувати графіки та

застосовувати складні дисплейні фільтри, що вимагає значних часових витрат.

3. Таким чином, класичні інструменти пасивного моніторингу класу Wireshark та tcpdump є ефективними засобами для детального аналізу мережевих проблем після того, як вони вже сталися, проте вони абсолютно непридатні для виконання ролі автономних систем превентивного виявлення мережевих загроз у реальному часі. [1]

1.2. Огляд систем моніторингу інфраструктури (Nagios, Zabbix)

Коли мова йде про контроль працездатності серверів, комутаторів та загального стану комп'ютерного парку підприємства, адміністратори найчастіше використовують системи інфраструктурного моніторингу. Найбільш популярними та перевіреними часом рішеннями у цій сфері є платформи Nagios та Zabbix. [2, 3]

Принцип роботи цих систем кардинально відрізняється від пакетних сніферів, які ми розглядали раніше. Замість того, щоб безперервно перехоплювати та розбирати кожен байт трафіку, інфраструктурні монітори працюють за методом періодичного опитування пристроїв. Для цього вони використовують базові мережеві інструменти. Наприклад, доступність сервера перевіряється звичайними ICMP-запитами, які більше відомі як команди ping. Щоб зняти показники навантаження з комутатора або маршрутизатора, використовується протокол SNMP. Також на цільові комп'ютери часто встановлюють спеціальні програми у вигляді агентів, які збирають дані зсередини операційної системи та відправляють їх на центральний сервер.

Nagios більше орієнтований на просту перевірку статусів. Його головна задача полягає у тому, щоб вчасно повідомити адміністратора, чи працює певний сервіс, чи він зупинився.[2] Натомість Zabbix є набагато складнішою та комплекснішою платформою. Він вміє не лише перевіряти статуси, але й збирати точні числові показники в єдину базу даних.[3] На основі цих цифр Zabbix будує

графіки навантаження і дозволяє налаштовувати гнучкі тригери. Тригер спрацьовує тоді, коли певний показник переходить допустиму межу, наприклад, якщо процесор завантажений на сто відсотків або на жорсткому диску закінчилося вільне місце.

Незважаючи на те, що такі системи ідеально підходять для контролю заліза, вони мають критичні обмеження у питаннях безпеки та аналізу мережевого трафіку. Використовувати їх для виявлення кібератак практично неможливо з кількох причин:

1. Відсутність глибокого аналізу пакетів. Системи інфраструктурного моніторингу не підтримують технологію DPI і взагалі не заглядають всередину мережевих пакетів. Вони бачать лише загальну статистику, наприклад, скільки мегабайт пройшло через порт комутатора за хвилину. Через це вони фізично не здатні помітити приховані загрози. Зловмисник може підмінювати MAC-адреси при атаці ARP Spoofing, підбирати паролі методом BruteForce або вивантажувати дані через DNS-тунелювання, а система моніторингу цього просто не побачить, бо загальний обсяг трафіку при цьому майже не зміниться. Навіть масове сканування портів залишиться непоміченим, оскільки воно ніяк не впливає на загальну доступність цільового сервера.
2. Велика затримка реакції. Zabbix та Nagios опитують обладнання не постійно, а з певним інтервалом. Зазвичай цей інтервал складає від однієї до п'яти хвилин. Для сучасних кібератак це просто величезний проміжок часу. Якщо в мережі почнеться інтенсивна DDoS-атака, вона здатна повністю паралізувати роботу офісу за лічені секунди. Система моніторингу надішле сповіщення лише через кілька хвилин під час чергового опитування, коли мережа вже повністю перестане відповідати.
3. Складність розгортання та залежність від серверів. Щоб повноцінно запустити такий моніторинг, компанії потрібно виділити окремий сервер,

налаштувати бази даних і витратити купу часу на прописування шаблонів. Для невеликих офісів це часто буває занадто дорого і складно.

4. Виходить, що Nagios та Zabbix чудово виконують свою роботу, коли потрібно слідкувати за здоров'ям обладнання, але вони залишають мережу абсолютно сліпою до специфічних кіберзагроз, бо не працюють із живим трафіком у реальному часі. [2,3]

1.3. Огляд комерційних SIEM-систем (Splunk, IBM QRadar)

Найбільш технологічним та комплексним класом програмних рішень для моніторингу подій безпеки у сучасних телекомунікаційних мережах виступають платформи формату SIEM. Ця абревіатура розшифровується як система керування інформацією та подіями безпеки. Провідними представниками цього сегмента ринку, які найчастіше розгортають у великих корпоративних інфраструктурах, є системи Splunk та IBM QRadar.[4,5]

Принцип функціонування таких платформ кардинально відрізняється від звичайного перехоплення трафіку. Вони працюють за методом наскрізного збору та глибокої агрегації лог-файлів з абсолютно різних джерел інфраструктури. Системи SIEM безперервно збирають системні журнали з кінцевих робочих станцій, комутаторів ядра, маршрутизаторів, міжмережевих екранів та серверів баз даних. Після цього спеціальні програмні компоненти нормалізують усі ці різнорідні дані та зводять їх у єдину структуровану таблицю для зручного пошуку.

Головною перевагою комерційних платформ є вбудований механізм інтелектуальної кореляції подій. Математичне ядро системи здатне самостійно співставляти розрізнені факти з різних пристроїв і знаходити між ними логічний зв'язок. Наприклад, якщо платформа бачить кілька невдалих спроб входу на сервер з певної IP-адреси, а через хвилину фіксує сканування відкритих портів з тієї ж адреси, вона автоматично об'єднує ці події в один критичний інцидент

безпеки. Це дозволяє вчасно виявляти складні та розподілені кібератаки, які звичайні сніфери чи монітори інфраструктури просто не помітять.

Проте практичне впровадження платформ класу Splunk або IBM QRadar для захисту локальних мереж малого та середнього бізнесу стикається із серйозними бар'єрами. Для компаній з мережею на кілька десятків робочих станцій подібні рішення є економічно та технічно невиправданими.[4,5] Використання комерційних SIEM-систем супроводжується трьома критичними недоліками:

По-перше, ці платформи висувають екстремально високі вимоги до апаратної інфраструктури. Для стабільної роботи алгоритмів кореляції та швидкого пошуку по базах даних необхідно розгорнути виділений кластер серверів із дуже швидкими дисковими масивами та великим об'ємом оперативної пам'яті. Запустити такий комплекс на звичайному персональному комп'ютері або базовому офісному сервері технічно неможливо.

По-друге, головним обмежуючим фактором виступає фінансова недоступність. Комерційні рішення вимагають регулярної оплати ліцензій, вартість яких стартує від кількох тисяч доларів на рік. Така ціна є абсолютно непідйомною для підприємств середнього масштабу, де часто немає окремих великих бюджетів на побудову центрів кібербезпеки.

По-третє, вагомим бар'єром є складність налаштування та подальшої технічної підтримки. Системи цього класу не працюють самостійно відразу після встановлення. Для їхньої правильної роботи компанія повинна мати в штаті вузькопрофільних спеціалістів, які будуть постійно писати нові правила аналізу подій та адаптувати систему під специфіку конкретної мережі.

Отже, комерційні SIEM-системи ідеально вирішують задачу комплексного моніторингу безпеки та автоматичного виявлення загроз, але вони вимагають колосальних бюджетів та розгортання окремої важкої серверної інфраструктури. Це залишає відкритою потребу в розробці легковагових програмних методів, які б

могли автоматично аналізувати трафік у реальному часі та працювати на стандартному обладнанні без великих капіталовкладень.

1.4. Порівняльний аналіз

Проведений огляд трьох категорій інструментів виявив, що кожна з них вирішує лише частину задачі автоматизованого захисту мережі. Для систематизації отриманих результатів наведено порівняльний аналіз за ключовими технічними характеристиками (таблиця 1.1).[1,2,3,4,5]

Таблиця 1.1 – Порівняльний аналіз інструментів моніторингу мережевого трафіку

Критерій	Wireshark / tcpdump	Nagios / Zabbix	Splunk / QRadar	TrafficAnalyzer
Метод захоплення	libpcap, promiscuous mode	ICMP, SNMP, агент	Агрегація логів	Scapy, L2–L7
Метод детекції загроз	Відсутній	Порогові значення SNMP	Кореляція подій	Статистика + евристика + ентропія
Класи загроз	Немає	Перевантаження каналу	Залежить від конфігурації	5 класів (DDoS, PortScan, ARP, BruteForce, DNS)
Аналіз пакетів (DPI)	Так	Ні	Частково	Так, від рівня L2 до L7
Метрики QoS	Обмежено	Latency, uptime	Залежить від джерел	Bandwidth, RTT, Packet Loss
Час реакції	Ручний аналіз	1–5 хвилин	Залежить від конфігурації	менше 1 с
Серверна інфраструктура	Не потрібна	Потрібна	Потрібна	Не потрібна

З таблиці 1.1 видно, що жоден з розглянутих інструментів не поєднує одночасно автоматичну детекцію кількох класів загроз, глибокий аналіз пакетів і розгортання без додаткової серверної інфраструктури.

Пасивні сніфери надають повну картину трафіку на рівнях L2-L7, але повністю залежать від людини при аналізі результатів. Wireshark і tcpdump є незамінними інструментами для ретроспективної діагностики конкретного інциденту, проте не підходять для постійного автоматизованого моніторингу: адміністратор не може цілодобово стежити за потоком сирих пакетів.

Системи інфраструктурного моніторингу Nagios і Zabbix автоматизують контроль стану обладнання, але залишають мережу сліпою до загроз, які не впливають на доступність вузлів. Атака типу PortScan взагалі не змінює показники доступності цільового сервера, тому система опитування не помітить її аж до наступного планового циклу.

Комерційні SIEM-платформи технічно здатні вирішити задачу комплексного захисту, але їхнє впровадження вимагає виділеної серверної інфраструктури та ліцензійних витрат, які є економічно невиправданими для підприємств середнього масштабу.

Таким чином, аналіз підтверджує наявність незаповненої ніші: інструмент, який поєднує автоматичну детекцію кількох класів загроз, глибокий аналіз пакетів і пасивне вимірювання метрик QoS, при цьому розгортається на стандартному ПК без додаткових витрат. Саме таким інструментом є розроблений у рамках даної роботи програмний комплекс TrafficAnalyzer, детальний опис методів якого наведено у наступному розділі.

Висновок до розділу 1

1. У розділі проведено аналіз трьох категорій існуючих рішень для моніторингу мережевого трафіку. Встановлено, що пасивні сніфери (Wireshark, tcpdump) забезпечують глибокий аналіз пакетів, але позбавлені механізмів автоматичної детекції загроз. [1]

2. Системи інфраструктурного моніторингу (Nagios, Zabbix) автоматизують контроль обладнання, але не виконують аналіз вмісту пакетів і мають затримку реакції від однієї до п'яти хвилин. [2, 3] Комерційні SIEM-платформи (Splunk, IBM QRadar) вирішують задачу комплексно, проте вимагають виділеної серверної інфраструктури та значних фінансових витрат, що робить їх недоступними для малого і середнього бізнесу. [4, 5]
3. Виявлена незаповнена ніша обумовлює необхідність розробки легковагового програмного методу, здатного автоматично аналізувати трафік у реальному часі на стандартному обладнанні. [7, 8, 9]

РОЗДІЛ 2
МЕТОД АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ МЕРЕЖЕВИХ АНОМАЛІЙ

2.1. Математична модель системи моніторингу та гібридного аналізу трафіку

Для формалізації задачі дослідження та математичного обґрунтування роботи розробленого програмного комплексу необхідно побудувати його структурно-параметричну модель. Процес моніторингу та детекції загроз у системі TrafficAnalyzer можна представити у вигляді взаємодії вхідних даних, внутрішніх налаштувань та результатів аналізу. [10, 11]

Математична модель такої системи описується за допомогою наступного кортежу множин [7, 8]:

$$S = (X, U, Y, F) \qquad (2.1)$$

Де:

- S– загальна система автоматизованого аналізу трафіку;
- X – множина вхідних параметрів (характеристики перехоплених мережевих пакетів);
- U – множина керуючих параметрів (внутрішні налаштування алгоритмів та порогові значення);
- Y – множина вихідних параметрів (результати детекції та метрики якості мережі);
- F – функціональний оператор переходу (алгоритми прийняття рішень).

Для детального розуміння логіки роботи системи необхідно розкрити структуру кожної з наведених множин.

Множина вхідних параметрів формується модулем захоплення пакетів (сніфером) і містить ключові ознаки кожного мережевого з'єднання. Вона має наступний вигляд:

$$X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\} \qquad (2.2)$$

Де:

- x_1 - IP-адреса джерела трафіку;
- x_2 - IP-адреса призначення;
- x_3 та x_4 - номери портів джерела та цільового вузла відповідно;
- x_5 - комбінація керуючих TCP-прапорів (SYN, ACK, FIN, RST);
- x_6 - загальний розмір корисного навантаження пакета у байтах;
- x_7 - точна часова мітка прибуття пакета.

Множина керуючих параметрів визначає межі чутливості системи. Ці значення задаються адміністратором для адаптації методу під конкретну мережу:

$$U = \{u_1, u_2, u_3\} \quad (2.3)$$

Де:

- u_1 - порогове значення інтенсивності запитів для детекції об'ємних атак;
- u_2 - часовий розмір ковзного вікна у секундах, протягом якого система накопичує статистику;
- u_3 - максимально допустима кількість невдалих спроб автентифікації для спрацювання алгоритму захисту від перебору паролів.

Множина вихідних параметрів формується після завершення аналізу і передається у веб-інтерфейс системи. Вона поділяється на індикатори безпеки та метрики якості обслуговування:

$$Y = \{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8\} \quad (2.4)$$

Де:

- $y_1 - y_5$ - бінарні індикатори виявлення п'яти базових загроз (DDoS, PortScan, ARP Spoofing, BruteForce, DNS Tunneling), при цьому значення 1 означає наявність атаки, а 0 вказує на нормальний стан;
- y_6 - показник кругової затримки пакета (RTT), виміряний у мілісекундах;
- y_7 - відсоток втрачених пакетів (Packet Loss);
- y_8 - загальна пропускна здатність поточного з'єднання (Bandwidth).

Функціональний оператор здійснює математичне перетворення вхідних даних і налаштувань у кінцевий результат. Оскільки система використовує гібридний метод, цей оператор розпадається на дві складові. Перша складова F_{stat} є статистичною і працює з вікном часу для пошуку DDoS та BruteForce. Друга складова F_{heur} є евристичною і перевіряє структуру пакета для виявлення PortScan, ARP Spoofing та DNS Tunneling. Загальний зв'язок параметрів описується виразом:

$$Y = F(X, U) = F_{stat}(X, U) \cup F_{heur}(X, U) \quad (2.5)$$

Така структурно-параметрична модель дозволяє чітко розмежувати процес збору мережевої статистики від процесу виявлення аномалій. Це слугує прямим математичним фундаментом для подальшої програмної реалізації комплексу на мові програмування Python.

2.2. Статистичні та інформаційно-ентропійні методи детекції аномалій (DDoS, DNS Tunneling)

Згідно з побудованою математичною моделлю, першою складовою загального алгоритму детекції є функція статистичного аналізу. Вона відповідає за виявлення тих кіберзагроз, які неможливо ідентифікувати за одним окремим пакетом. До таких загроз у нашому комплексі належать розподілені атаки на відмову в обслуговуванні (DDoS) та використання прихованих каналів зв'язку (DNS Tunneling). Для їхнього виявлення система використовує два різні математичні підходи: аналіз ковзного вікна та обчислення ентропії.

Перший підхід базується на статистичному аналізі відхилень і застосовується виключно для виявлення DDoS-атак. Логіка його роботи полягає у тому, що система постійно відстежує загальну кількість пакетів, які надходять з кожної унікальної IP-адреси у межах заданого ковзного вікна часу. У нормальному режимі роботи мережі це значення є відносно стабільним і не має різких стрибків.

Для оцінки інтенсивності вхідного потоку програма розраховує швидкість надходження запитів за такою формулою [12, 13]:

$$R = \frac{N}{\Delta t} \tag{2.6}$$

Де:

- R - швидкість надходження пакетів від конкретного джерела (кількість запитів за секунду);
- N - загальна кількість перехоплених пакетів від однієї IP-адреси;
- Δt - тривалість ковзного вікна часу у секундах.

Як тільки значення швидкості для певного джерела різко перевищує безпечне середнє значення по вікну (порогове значення u_1 з моделі), система автоматично фіксує аномалію і генерує сповіщення. Простіше кажучи, математичний апарат запам'ятовує базову лінію поведінки мережі і миттєво реагує, коли трафік від неї відхиляється. Це дозволяє ефективно блокувати флуд-атаки на ранніх стадіях.

Другий підхід є більш складним і базується на обчисленні ентропії Шеннона. Цей алгоритм використовується для виявлення загрози DNS Tunneling. [14, 15, 16] Ентропія у теорії інформації виступає математичною мірою хаотичності або непередбачуваності даних.

Нормальний легітимний DNS-запит має цілком передбачувану структуру і низьку ентропію (наприклад, google.com). Натомість тунельований запит, який використовують зловмисники для прихованої передачі даних, містить зашифровану інформацію і виглядає як абсолютно випадковий набір символів (наприклад, xK93mZpqL7nVbR2w.evil.com). Через велику кількість унікальних літер та цифр такий запит має екстремально високу ентропію.

Система розраховує ентропію для кожного вхідного DNS-запиту за класичною формулою [14]:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_2 P(x_i) \tag{2.7}$$

Де:

- $H(X)$ - значення інформаційної ентропії послідовності символів;
- n - кількість унікальних символів у доменному імені;
- $P(x_i)$ - ймовірність появи конкретного символу в цьому рядку;
- log_2 - логарифм за основою два, який визначає одиницю виміру ентропії у бітах.

Після обчислення система порівнює результат із заданими порогами. Практичні тести показують, що якщо значення ентропії перевищує 3,5 біта на символ, або якщо загальна довжина запиту становить понад 50 символів, комплекс автоматично фіксує аномалію та сигналізує про спробу використання DNS-тунелю. Такий математичний критерій дозволяє уникнути хибних спрацьовувань на звичайних довгих адресах. [16]

2.3. Евристичні методи ідентифікації структурних аномалій (PortScan, BruteForce, ARP Spoofing)

Другою складовою гібридного методу детекції є функція евристичного аналізу. Вона застосовується для ідентифікації тих загроз, які мають чітко виражену структуру або специфічний алгоритм виконання. До цього класу атак у системі TrafficAnalyzer належать сканування портів (PortScan), перебір паролів (BruteForce) та підміна фізичних адрес (ARP Spoofing). [17, 18]

Логіка роботи евристичного аналізатора кардинально відрізняється від статистичного підходу. У цьому випадку система не шукає відхилень від загальної норми трафіку. Натомість вона використовує порогові функції та лічильники, які фіксують конкретні підозрілі події у заданому часовому вікні.

Для виявлення розвідувального сканування портів система аналізує кількість унікальних цільових портів, до яких намагається підключитися одна IP-адреса за короткий проміжок часу. Математично ця умова перевірки виглядає наступним чином [19, 20]:

$$C_{ports}(IP_{src}, \Delta t) \geq U_{scan} \qquad (2.8)$$

Де:

- C_{ports} - кількість унікальних цільових портів, до яких зафіксовано спробу підключення;
- IP_{src} - IP-адреса потенційного зловмисника;
- Δt - часове вікно аналізу;
- U_{scan} - порогове значення допустимої кількості портів.

Якщо лічильник унікальних портів перевищує заданий поріг, програма миттєво генерує сповіщення про загрозу PortScan.

Схожий пороговий механізм застосовується для захисту від атак типу BruteForce. Проте в цьому випадку система рахує не кількість портів, а кількість невдалих спроб встановлення TCP-з'єднання або авторизації з однієї адреси на один конкретний цільовий порт (наприклад, порт 22 для SSH або 3389 для RDP). Формула спрацювання має такий вигляд [19]:

$$C_{fails}(IP_{src}, Port_{dst}\Delta t) \geq U_{brute} \quad (2.9)$$

Де:

- C_{fails} - лічильник невдалих спроб підключення;
- $Port_{dst}$ - цільовий порт, який піддається атаці;
- U_{brute} - максимально допустима кількість невдалих спроб (ліміт).

Третій евристичний алгоритм захищає мережу від підміни адрес (ARP Spoofing). Він не використовує часові вікна, а працює за принципом миттєвої перевірки структурної цілісності. Система витягує з кожного вхідного ARP-пакета пару адрес (IP та MAC) і порівнює її з еталонними записами, які зберігаються у внутрішньому кеші таблиці маршрутизації хоста. Функція детекції спрацьовує за такої логічної умови [21]:

$$MAC_{packet}(IP_x) \neq MAC_{cahce}(IP_x) \quad (2.10)$$

Де:

- MAC_{packet} - фізична адреса, яка вказана у вхідному мережевому пакеті для певної IP-адреси;
- MAC_{cahce} - легітимна фізична адреса, яка була збережена у довіреній таблиці системи для цієї ж IP-адреси;
- IP_x - IP-адреса конкретного мережевого вузла, для якого здійснюється перевірка відповідності MAC-адреси.

Як тільки система фіксує невідповідність MAC-адреси у пакеті із збереженим кешем, вона розуміє, що зловмисник намагається перенаправити трафік на свій вузол, і одразу сигналізує про аномалію.

Таким чином, три евристичні детектори використовують окремі математичні правила під кожну загрозу. Це забезпечує перевірку трафіку в реальному часі без перевантаження системи.

2.4. Математичний апарат пасивного обчислення метрик якості обслуговування (QoS)

Четвертою, але не менш важливою складовою програмного комплексу виступає модуль контролю якості обслуговування мережі (QoS). Більшість класичних систем моніторингу використовують активні методи вимірювання. Вони генерують спеціальні тестові пакети (наприклад, ICMP-запити) і постійно відправляють їх у мережу, щоб перевірити затримку або швидкість. [22, 11]Такий підхід створює додаткове штучне навантаження на комутатори і забиває канали зв'язку зайвим трафіком.

У розробленому методі реалізовано принципово інший підхід. Система здійснює виключно пасивні вимірювання. Вона не відправляє жодного власного байта даних, а розраховує всі необхідні метрики прямо на льоту, математично аналізуючи службові поля тих пакетів, які вже проходять через мережевий інтерфейс.

Першим базовим показником є загальна пропускна здатність каналу (Bandwidth). Для її розрахунку система підсумовує розмір усіх корисних навантажень пакетів за певне часове вікно. Формула обчислення має такий вигляд [10]:

$$B = \frac{1}{\Delta t} \sum_{i=1}^{N} L_i \quad (2.11)$$

Де:

- B - поточна пропускна здатність каналу зв'язку (Bandwidth) у байтах за секунду;
- N - загальна кількість пакетів, зафіксованих у межах вікна;
- L_i - загальний розмір i - пакету у байтах;
- Δt - тривалість інтервалу вимірювання у секундах.

Другим критично важливим показником є кругова затримка (Round-Trip Time, або RTT). У пасивному режимі цей показник вираховується завдяки особливостям роботи протоколу TCP. Коли комп'ютер відправляє пакет із даними, приймаюча сторона зобов'язана надіслати у відповідь службовий пакет-підтвердження (прапор ACK). Програма фіксує точний час відправки оригінального пакета і час отримання відповідного підтвердження. Середня затримка для з'єднання розраховується за такою формулою [10,11]:

$$RTT_{avg} = \frac{1}{M} \sum_{i=1}^{M} t_{ACK,i} - t_{DATA,i} \quad (2.12)$$

Де:

- RTT_{avg}- середній час кругового обходу пакета у мілісекундах;
- M - кількість успішно підтверджених пар пакетів;
- $t_{DATA,i}$ - часова мітка фіксації відправленого пакета з даними;
- $t_{ACK,i}$ - часова мітка фіксації відповідного пакета-підтвердження від сервера.

Третім показником якості виступає відсоток втрачених пакетів (Packet Loss). У пасивному режимі система не має можливості знати напевно, скільки пакетів було загублено десь у глобальній мережі інтернет. Проте вона здатна фіксувати повторні передачі (TCP Retransmissions). Якщо пакет губиться по дорозі, відправник через певний час автоматично надсилає його точну копію. Програма рахує співвідношення таких вимушених дублікатів до загального обсягу трафіку за формулою [10]:

$$L = \left(\frac{N_{retrans}}{N_{total}}\right) * 100\% \quad (2.13)$$

Де:

- L - відсоток втрачених пакетів у мережі;
- $N_{retrans}$ - кількість пакетів, які були класифіковані як повторна передача даних;
- N_{total} - загальна кількість зафіксованих пакетів для поточного з'єднання.

Такий математичний апарат дозволяє адміністратору отримувати точні дані про стан і завантаженість інфраструктури у реальному часі без генерації жодного тестового запиту, що робить моніторинг абсолютно непомітним для мережі.

2.5. Обґрунтування ефективності комплексного гібридного підходу

У попередніх підрозділах було детально розглянуто архітектуру та математичний апарат розробленого програмного комплексу. Спираючись на ці дані, можна стверджувати, що класичні одномодельні системи захисту більше не здатні ефективно протистояти сучасним загрозам. Саме тому в основу TrafficAnalyzer закладено комплексний гібридний підхід, який об'єднує чотири незалежні аналітичні механізми в єдину екосистему.

Головна перевага розробленого методу полягає у синергії його компонентів. Кожен окремий модуль перекриває недоліки іншого.

Статистичний аналізатор на базі ковзного вікна успішно закриває вразливість інфраструктури до масових об'ємних атак (DDoS). Він дозволяє системі динамічно адаптуватися до нормального фонового навантаження мережі і реагувати лише на критичні відхилення. Водночас інформаційно-ентропійний алгоритм вирішує проблему прихованих загроз. Він здатний математично розпізнати зашифрований трафік у, здавалося б, легітимних DNS-запитах (DNS Tunneling), які звичайні міжмережеві екрани пропускають без перевірки.

Оскільки статистика не працює для точкових втручань, третім рівнем захисту виступає евристичний модуль. Завдяки системі жорстких порогових лічильників та миттєвій перевірці структурної цілісності пакетів, він забезпечує блискавичну детекцію сканування портів (PortScan), спроб перебору паролів (BruteForce) та підміни маршрутизації (ARP Spoofing).

Інтеграція модуля пасивного вимірювання показників якості обслуговування (QoS) робить розроблений підхід унікальним серед аналогів легкого класу. Завдяки математичному аналізу службових прапорів TCP, система здатна в реальному часі контролювати пропускну здатність, затримки (RTT) та втрати пакетів. При цьому програма не генерує жодного власного тестового запиту, що зберігає пропускну здатність каналів зв'язку виключно для корисного трафіку користувачів.

Якщо порівнювати розроблений гібридний метод із системами, які розглядалися у першому розділі, можна виділити кілька ключових здобутків. На відміну від пасивних сніферів (Wireshark), TrafficAnalyzer приймає рішення автоматично і не вимагає постійної уваги адміністратора. [28] На відміну від інфраструктурних моніторів (Zabbix), він заглядає всередину пакетів і реагує на загрози за частки секунди, а не чекає наступного циклу опитування. [2,3] І на відміну від комерційних SIEM-систем, він не потребує розгортання дорогих виділених серверів або кластерів баз даних. [4,5]

Таким чином, запропонована структурно-параметрична модель та набір математичних алгоритмів є повністю обґрунтованими. Вони формують надійну теоретичну базу для переходу до наступного етапу роботи, а саме до практичної

реалізації цього гібридного методу у вигляді програмного коду на мові Python з використанням бібліотеки Scapy. [23, 24]

Висновок до розділу 2

1. У розділі розроблено математичний апарат гібридного методу автоматизованого виявлення мережевих аномалій. Побудована структурно-параметрична модель системи у вигляді кортежу S=(X,U,Y,F) формалізує процес перетворення вхідних характеристик мережевих пакетів у результати детекції та метрики якості обслуговування.
2. Статистична складова методу на основі ковзного вікна часу забезпечує виявлення об'ємних атак типу DDoS шляхом порівняння поточної інтенсивності трафіку з базовою лінією мережі. Інформаційно-ентропійний алгоритм Шеннона дозволяє виявляти приховані DNS-канали за структурою доменних імен без аналізу корисного навантаження пакета.
3. Евристична складова реалізує три незалежні детектори: пороговий лічильник унікальних портів для виявлення PortScan, лічильник невдалих TCP-з'єднань для виявлення BruteForce та перевірку цілісності ARP-таблиці для виявлення підміни фізичних адрес.
4. Модуль пасивного вимірювання показників QoS розраховує пропускну здатність, кругову затримку та відсоток втрачених пакетів виключно на основі службових полів перехопленого трафіку, без генерації тестових запитів. Розроблений математичний апарат є повною теоретичною основою для програмної реалізації комплексу TrafficAnalyzer, яка описана у наступному розділі.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ TRAFFICANALYZER

3.1. Обґрунтування вибору технологічного стеку

Вибір інструментів для реалізації програмного комплексу визначається технічними вимогами до системи: пасивне захоплення пакетів на рівнях L2–L7, автоматична детекція аномалій у реальному часі, відображення результатів у веб-інтерфейсі без генерації тестового трафіку та розгортання на стандартному ПК без додаткової серверної інфраструктури. Для кожного рівня конвеєру обробки даних розглянуто кілька альтернатив і обрано оптимальний інструмент.

3.1.1 Вибір мови програмування

Основною мовою реалізації обрано Python 3. Головним аргументом є наявність бібліотеки Scapy, яка забезпечує нативний розбір пакетів на всіх рівнях моделі OSI без написання власного парсера байтів. [23,24] Альтернативні мови такі як Go та C++ забезпечують вищу продуктивність, проте не мають еквівалентної екосистеми для роботи з мережевими пакетами на рівні готових об'єктів з розібраними полями. Для моніторингу офісної мережі зі швидкістю до 1 Gbps продуктивності Python цілком достатньо, а перевага у швидкості розробки і читабельності коду є визначальною для прикладного інженерного рішення.

3.1.2 Вибір бібліотеки захоплення пакетів

Вибір інструменту захоплення пакетів є ключовим архітектурним рішенням системи. У таблиці 3.1 наведено порівняння основних альтернатив.

Таблиця 3.1 – Порівняльний аналіз бібліотек захоплення мережевих пакетів

Таблиця 3.1 – Порівняльний аналіз бібліотек захоплення мережевих пакетів

Критерій	libpcap / Npcap	pyshark	dpdk	Scapy
Рівень захоплення	L2	L2	L2	L2–L7
Розбір заголовків	Сирі байти	Через tshark	Сирі байти	Нативний

Доступ до полів пакета	Ручний парсинг	pkt.ip.src	Ручний парсинг	pkt[IP].src
Аналіз ARP-пакетів	Потребує парсингу	Так	Потребує парсингу	pkt[ARP].op
Затримка обробки	Мінімальна	Висока (tshark)	Мінімальна	Низька
Залежність від зовнішніх процесів	Ні	Так (tshark)	Ні	Ні
Придатність для пасивного RTT	Ні	Частково	Ні	Так

З таблиці 3.1 видно що Scapy є єдиним інструментом серед розглянутих, який забезпечує нативний розбір пакетів на всіх рівнях без залежності від зовнішніх процесів. [23] Термін "нативний розбір" означає що Scapy самостійно розпізнає протокол кожного пакета і одразу надає доступ до його полів через зручний програмний інтерфейс. Наприклад, для отримання IP-адреси джерела достатньо звернутись до поля pkt[IP].src, для аналізу ARP-пакетів використовується поле pkt[ARP].op, для перевірки TCP-прапорів застосовується pkt[TCP].flags. Бібліотеки libpcap і dpdk такого інтерфейсу не надають і повертають лише масив байтів, який розробник змушений розбирати вручну для кожного протоколу окремо. Бібліотека pyshark запускає окремий процес утиліти tshark для кожного сеансу захоплення, що вносить значну затримку і унеможливлює реакцію системи менше однієї секунди. [1] У реалізованому комплексі виклик scapy.sniff(iface=ifaces, prn=callback, store=False) забезпечує безперервне захоплення з передачею кожного пакета у вигляді готового об'єкта з розібраними полями безпосередньо в обробник.

3.1.3 Вибір веб-фреймворку та механізму push-сповіщень

Для реалізації серверної частини веб-інтерфейсу розглянуто три варіанти, наведені у таблиці 3.2.

Таблиця 3.2 – Порівняльний аналіз веб-фреймворків

Критерій	Django	FastAPI	Flask + Socket.IO
Складність налаштування	Висока	Середня	Низька
Push-сповіщення	Потребує додаткових рішень	SSE або WebSocket	WebSocket нативно
Режим роботи	Sync / Async	Async	Threading
Затримка від події до браузера	Залежить від конфігурації	Низька	Менше 1 с
Доцільність для даної задачі	Надлишковий	Вимагає async скрізь	Оптимальний

Flask обрано через просту інтеграцію з бібліотекою Flask-SocketIO, яка реалізує WebSocket-з'єднання у threading-режимі. [25,26] Це дозволяє серверу самостійно надсилати події браузеру у момент виявлення загрози через виклик socketio.emit("threat", event) без очікування запиту від клієнта. FastAPI вимагає використання конструкцій async/await у всьому коді, що ускладнює інтеграцію з синхронним сніфером Scapy і збільшує складність підтримки коду. [26] Django є надлишковим рішенням для задачі реального часу оскільки орієнтований на класичні веб-додатки з рендерингом сторінок і не має вбудованої підтримки WebSocket.

3.1.4 Вибір інструменту візуалізації

Для відображення живих графіків у браузері розглянуто три варіанти, наведені у таблиці 3.3.

Таблиця 3.3 – Порівняльний аналіз інструментів візуалізації

Критерій	Grafana	D3.js	Chart.js
Потреба у базі даних	Так (Prometheus/InfluxDB)	Ні	Ні
Окремий сервер	Так	Ні	Ні
Складність інтеграції	Висока	Висока	Низька
Живі графіки через WebSocket	Потребує налаштування	Вручну	Нативно
Придатність без інфраструктури	Ні	Так	Так

Grafana потребує розгортання окремої бази даних часових рядів такої як Prometheus або InfluxDB, а також додаткового сервера, що суперечить вимозі розгортання на стандартному ПК. D3.js є потужним інструментом побудови інтерактивних графіків, проте вимагає значного обсягу коду навіть для простого живого графіка оскільки є низькорівневою бібліотекою без готових компонентів. Chart.js вбудовується безпосередньо в HTML-сторінку і оновлюється через WebSocket-події без додаткових залежностей, що робить його оптимальним вибором для даної задачі. [27]

3.1.5 Конфігурація системи

Усі порогові значення алгоритмів детекції винесено в окремий файл config.py, що дозволяє адміністратору адаптувати систему під конкретну мережу без зміни вихідного коду. Параметр DDOS_WINDOW_SIZE = 30 визначає розмір ковзного вікна у секундах. Параметр DDOS_THRESHOLD_MULTIPLIER = 10 задає коефіцієнт відхилення від середнього для спрацювання детектора. Параметр DDOS_MIN_PACKETS = 2000 встановлює мінімальний поріг абсолютної кількості пакетів для уникнення хибних спрацьовувань на малоактивних мережах. Така архітектура конфігурації відповідає принципу

розділення відповідальності і спрощує адаптацію системи під різні умови експлуатації.

3.2. Архітектура програмного комплексу TrafficAnalyzer

Програмний комплекс TrafficAnalyzer побудовано за принципом багаторівневого конвеєру обробки даних. [18,28] Кожен рівень конвеєру виконує єдину чітко визначену функцію і взаємодіє із сусідніми рівнями через фіксований інтерфейс. Такий підхід забезпечує незалежність модулів між собою: заміна або розширення будь-якого компонента не вимагає змін у решті коду системи. Загальну структурну схему конвеєру наведено на рисунку 3.1.

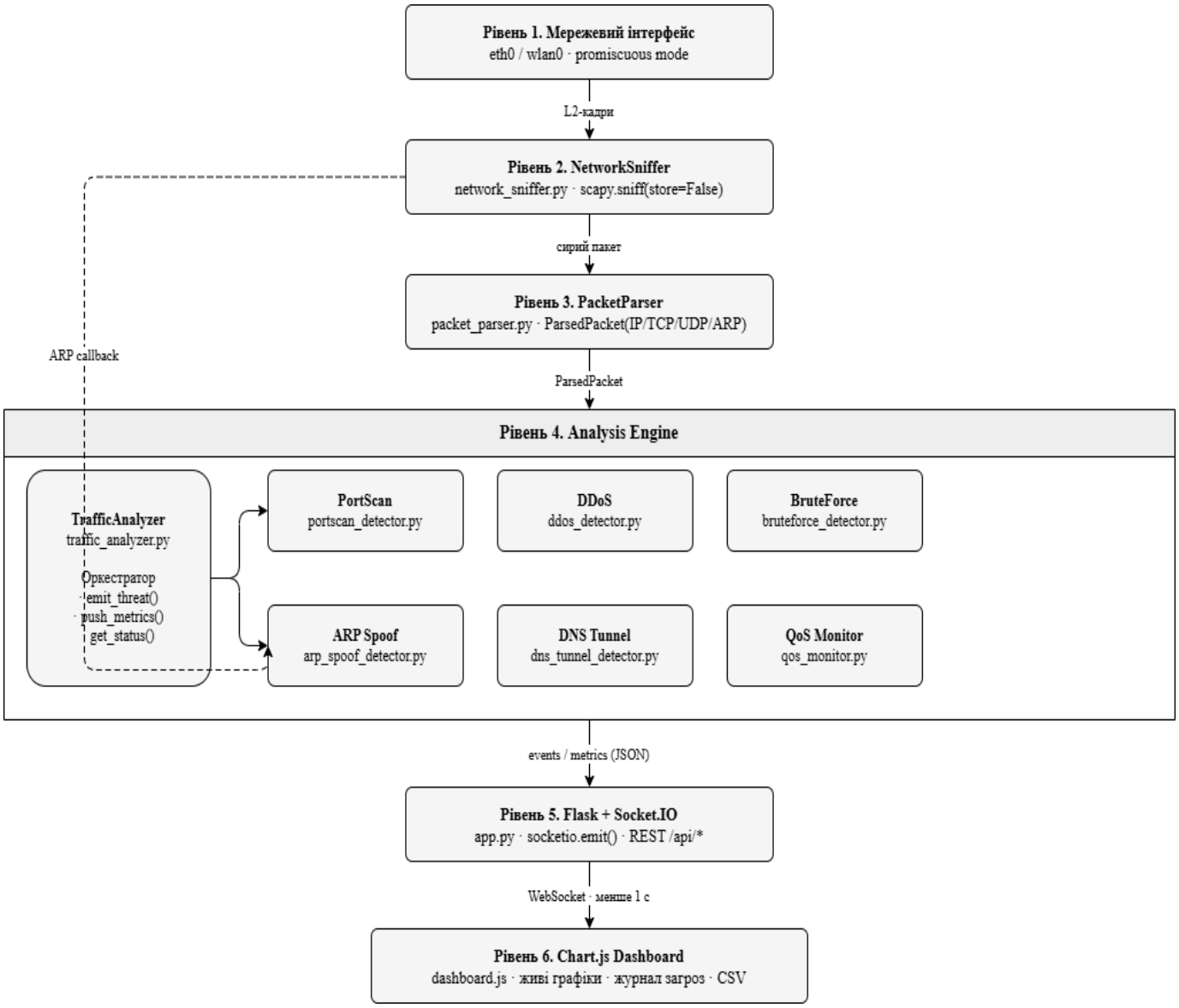

Рисунок 3.1 – Структурна схема конвеєру обробки мережевого трафіку TrafficAnalyzer

Конвеєр складається з шести рівнів. Характеристики кожного рівня наведено у таблиці 3.4.

Таблиця 3.4 – Компоненти конвеєру обробки трафіку

Рівень	Компонент	Модуль	Функція
1	Мережевий інтерфейс	eth0 / wlan0	Захоплення L2-кадрів у promiscuous mode
2	NetworkSniffer	network_sniffer.py	Безперервне захоплення пакетів через scapy.sniff()
3	PacketParser	packet_parser.py	Розбір заголовків у структуру ParsedPacket
4	Analysis Engine	traffic_analyzer.py + 6 модулів	Детекція аномалій та розрахунок метрик QoS
5	Flask + Socket.IO	app.py	REST API та WebSocket push-сповіщення
6	Chart.js Dashboard	dashboard.js	Відображення живих графіків та журналу загроз

Рівень 2. захоплення пакетів

Захоплення мережевого трафіку реалізовано у класі NetworkSniffer файлу network_sniffer.py. При ініціалізації клас автоматично визначає доступні мережеві інтерфейси методом _detect_interfaces(), який фільтрує службові адаптери такі як петлевий інтерфейс, VMware та Bluetooth, і залишає лише реальні мережеві карти з дійсними IP-адресами. Якщо адміністратор явно вказав

інтерфейс через параметр NETWORK_INTERFACE у файлі config.py, автоматичне визначення пропускається.

Основний цикл захоплення реалізовано методом _sniff_loop():

```
def _sniff_loop(self):
        scapy.sniff(
         iface=self._ifaces if self._ifaces else None,
         filter=config.CAPTURE_FILTER,
         prn=self._on_packet,
         store=False,
         stop_filter=lambda _: self._stop_event.is_set(),
        )
```

Параметр store=False є критично важливим для тривалої роботи системи. Він забороняє Scapy накопичувати перехоплені пакети у внутрішньому списку, що повністю усуває ризик переповнення оперативної пам'яті. Без цього параметра при тривалому моніторингу активної мережі Scapy накопичив би мільйони пакетів у пам'яті і система аварійно завершила б роботу.

Параметр stop_filter перевіряє стан прапорця зупинки після кожного пакета, що дозволяє коректно завершити роботу сніфера без примусового знищення потоку. Параметр filter приймає BPF-вираз з файлу конфігурації і передає його безпосередньо в ядро операційної системи, де відбувається апаратна фільтрація до того як пакет потрапить у простір користувача. [6]

Для запобігання блокуванню головного потоку програми сніфер запускається у окремому фоновому потоці:

```
def start(self):
   self._stop_event.clear()
   self._thread = threading.Thread(
      target=self._sniff_loop, name="sniffer", daemon=True
   )
   self._thread.start()
```

Прапорець daemon=True гарантує що потік сніфера автоматично завершиться при закритті головного процесу Python навіть якщо адміністратор не викликав метод stop() явно. Між сніфером і модулями аналізу розміщено чергу пакетів з обмеженим розміром:

self._queue = queue.Queue(maxsize=10_000)

Якщо черга заповнена, найстаріший пакет видаляється і на його місце додається новий. Таке рішення забезпечує стійкість системи в умовах пікового навантаження: навіть при раптовому сплеску трафіку черга не блокує роботу сніфера і система продовжує отримувати нові пакети. Окрім черги, сніфер підтримує буфер сирих пакетів для подальшого експорту у форматі pcap:

self._raw_packets = deque(maxlen=5000)

Використання deque з обмеженою довжиною гарантує що буфер завжди містить останні 5000 пакетів і автоматично витісняє старіші при переповненні. Технічні характеристики модуля захоплення наведено у таблиці 3.5.

Таблиця 3.5 - Технічні характеристики модуля NetworkSniffer

Параметр	Значення
Бібліотека захоплення	Scapy 2.5+
Режим мережевого адаптера	promiscuous mode
Розмір черги пакетів	10 000 пакетів
Буфер сирих пакетів для pcap	5 000 пакетів (deque)
Фільтрація на рівні ядра	BPF через CAPTURE_FILTER
Запуск	Окремий daemon-потік
Зупинка	threading.Event()

Рівень 3. Розбір пакетів

Кожен перехоплений сирий пакет передається у функцію parse() модуля packet_parser.py. Ця функція аналізує структуру пакета, визначає протокол і витягує з заголовків усі поля необхідні для детекції аномалій. Результатом роботи функції є уніфікований об'єкт типу ParsedPacket.

Така уніфікація є важливим архітектурним рішенням: усі детектори вищого рівня працюють виключно з об'єктом ParsedPacket і не мають прямого доступу до сирих байтів пакета. Це ізолює деталі реалізації Scapy від логіки детекції і дозволяє тестувати кожен детектор окремо шляхом передачі йому підготовленого

об'єкта без реального трафіку. Структуру об'єкта ParsedPacket наведено у таблиці 3.6.

Таблиця 3.6 – Поля об'єкта ParsedPacket

Поле	Тип	Джерело в Scapy	Опис
timestamp	float	raw.time	Мітка часу захоплення пакета
src_ip	str	pkt[IP].src	IP-адреса джерела
dst_ip	str	pkt[IP].dst	IP-адреса призначення
protocol	str	визначається парсером	TCP / UDP / ICMP / ARP / HTTP / SSH та інші
size	int	len(raw)	Розмір пакета у байтах
src_port	int	pkt[TCP/UDP].sport	Порт джерела
dst_port	int	pkt[TCP/UDP].dport	Порт призначення
flag_syn	bool	TCP flags & 0x02	Прапорець SYN
flag_ack	bool	TCP flags & 0x10	Прапорець ACK
flag_fin	bool	TCP flags & 0x01	Прапорець FIN
flag_rst	bool	TCP flags & 0x04	Прапорець RST
tcp_seq	int	pkt[TCP].seq	Порядковий номер TCP
tcp_ack_num	int	pkt[TCP].ack	Номер підтвердження TCP
http_method	str	TCP payload	HTTP-метод, якщо виявлено

Логіка роботи функції parse() будується на послідовній перевірці шарів пакета:

```
def parse(raw: Packet) -> Optional[ParsedPacket]:
    if raw.haslayer(ARP):
```

```
        return _parse_arp(raw)
    if not raw.haslayer(IP):
        return None
    ip = raw[IP]
    pkt = ParsedPacket(
        timestamp=float(raw.time),
        src_ip=ip.src,
        dst_ip=ip.dst,
        size=len(raw)
    )
    if raw.haslayer(TCP):
        _parse_tcp(raw[TCP], pkt)
    elif raw.haslayer(UDP):
        _parse_udp(raw[UDP], pkt)
    elif raw.haslayer(ICMP):
        pkt.protocol = "ICMP"
    else:
        pkt.protocol = "OTHER"
    return pkt
```

ARP-пакети перевіряються першими оскільки вони не мають IP-шару і стандартна перевірка haslayer(IP) відкинула б їх. Якщо пакет не є ні ARP ні IP, функція повертає None і такий пакет ігнорується системою. Це відфільтровує службові кадри канального рівня такі як STP і LLDP які не несуть корисної інформації для аналізу безпеки.

Особливістю модуля є визначення протоколу прикладного рівня. Для TCP-пакетів система спочатку намагається визначити протокол за вмістом корисного навантаження, і лише якщо це не вдається, використовує таблицю відповідності портів:

```
def _parse_tcp(tcp, pkt):
    pkt.src_port = tcp.sport
    pkt.dst_port = tcp.dport
    pkt.tcp_seq = tcp.seq
    pkt.tcp_ack_num = tcp.ack
    flags = tcp.flags
    pkt.flag_syn = bool(flags & 0x02)
    pkt.flag_ack = bool(flags & 0x10)
    pkt.flag_fin = bool(flags & 0x01)
```

```
    pkt.flag_rst = bool(flags & 0x04)
    if tcp.dport == 80 or tcp.sport == 80:
        try:
            payload = bytes(tcp.payload)
            if payload[:4] in (b"GET ", b"POST", b"HEAD", b"PUT ", b"DELE"):
                pkt.protocol = "HTTP"
                pkt.http_method = payload.split(b" ")[0].decode(errors="ignore")
                return
        except Exception:
            pass
    pkt.protocol = (
        _TCP_PORT_PROTOCOLS.get(tcp.dport)
        or _TCP_PORT_PROTOCOLS.get(tcp.sport)
        or "TCP"
    )
```

Визначення TCP-прапорців реалізовано через побітові маски. Наприклад, вираз bool(flags & 0x02) виділяє другий біт байта прапорців, який відповідає прапорцю SYN. Такий підхід є більш надійним ніж порівняння рядкового представлення прапорців, оскільки не залежить від порядку їх відображення у Scapy. Перелік протоколів прикладного рівня що розпізнаються системою наведено у таблиці 3.7.

Таблиця 3.7 – Протоколи прикладного рівня що визначаються модулем PacketParser

Протокол	Порт TCP	Порт UDP	Метод визначення
HTTP	80, 8080	—	Аналіз payload + порт
HTTPS	443, 8443	—	Порт
SSH	22	—	Порт
FTP	20, 21	—	Порт
DNS	53	53	Порт
RDP	3389	—	Порт
SMB	445	—	Порт
DHCP	—	67, 68	Порт
NTP	—	123	Порт
SNMP	—	161, 162	Порт
MySQL	3306	—	Порт
Redis	6379	—	Порт
ARP	—	—	Окремий парсер

ICMP	—	—	Шар пакета

Рівень 4. Analysis Engine та оркестратор

Центральним елементом архітектури є клас TrafficAnalyzer у файлі traffic_analyzer.py, який виконує роль оркестратора конвеєру. Він ініціалізує всі шість аналітичних модулів, керує чергою пакетів та організовує їх послідовну обробку у безперервному циклі. Склад аналітичних модулів наведено у таблиці 3.8.

Таблиця 3.8 – Аналітичні модулі системи TrafficAnalyzer

Модуль	Файл	Клас загрози	Метод детекції
PortScanDetector	portscan_detector.py	PortScan	Лічильник унікальних портів
DDoSDetector	ddos_detector.py	DDoS	Ковзне вікно часу
BruteForceDetector	bruteforce_detector.py	BruteForce	Лічильник невдалих з'єднань
ARPSpoofDetector	arp_spoof_detector.py	ARP Spoofing	Перевірка кешу MAC-адрес
DNSTunnelDetector	dns_tunnel_detector.py	DNS Tunneling	Ентропія Шеннона
QoSMonitor	qos_monitor.py	—	Пасивний розрахунок метрик

Порядок виклику детекторів в основному циклі є фіксованим і має принципове значення:

```
def _loop(self):
    while self._running:
        pkt = self._sniffer.get_packet(timeout=0.05)
        if pkt:
            scan = self._portscan.process(pkt)
            if scan:
                self._emit_threat("PortScan", scan)
                self._ddos.mark_as_portscan(scan.src_ip)

            ddos = self._ddos.process(pkt)
```

```
        if ddos:
            self._emit_threat("DDoS", ddos)

        bf = self._bruteforce.process(pkt)
        if bf:
            self._emit_threat("BruteForce", bf)

        dt = self._dns_tunnel.process(pkt)
        if dt:
            self._emit_threat("DNS Tunneling", dt)

        self._qos.process(pkt)
    self._push_metrics()
```

PortScan перевіряється першим оскільки масове сканування портів генерує велику кількість коротких пакетів з однієї IP-адреси за короткий проміжок часу, що може хибно спрацювати як DDoS-атака. Якщо детектор PortScan ідентифікував певну IP-адресу як сканер, він передає цю інформацію детектору DDoS через виклик mark_as_portscan() і той ігнорує трафік з цієї адреси під час поточного вікна аналізу. Таке рішення усуває дублювання сповіщень про одну і ту ж загрозу.

Модуль ARP Spoofing отримує пакети окремим шляхом через прямий callback від сніфера до передачі пакета у чергу:

```
def _check_arp(self, raw):
    threat = self._arp_spoof.process_raw(raw)
    if threat:
        self._emit_threat("ARP Spoofing", threat)
```

Такий підхід обумовлений тим що перевірка ARP потребує доступу до сирого пакета на рівні L2 до його розбору у структуру ParsedPacket, оскільки детектор порівнює MAC-адресу з ARP-заголовка з кешем таблиці маршрутизації хоста.

Рівень 5. Серверна частина та передача подій

Серверну частину реалізовано у файлі app.py на основі фреймворку Flask з розширенням Flask-SocketIO. При виявленні загрози оркестратор викликає метод _emit_threat(), який записує подію у журнал і одразу надсилає її у браузер:

```
def _emit_threat(self, typ, t):
    entry = _thr(typ, t)
    self._threat_log.append(entry)
    if self._on_threat:
        self._on_threat(entry)
```

У файлі app.py цей callback пов'язаний з викликом socketio.emit("threat", e), що забезпечує миттєву доставку події всім підключеним клієнтам через WebSocket. Метрики QoS надсилаються окремо з фіксованим інтервалом в одну секунду через метод _push_metrics() незалежно від наявності загроз. Склад REST API серверної частини наведено у таблиці 3.9.

Таблиця 3.9 – REST API endpoints програмного комплексу

Endpoint	**Метод**	**Функція**
/	GET	Головна сторінка веб-інтерфейсу
/api/status	GET	Стан системи: running, queue_size, threats_total
/api/threats	GET	Повний журнал виявлених загроз у форматі JSON
/api/start	POST	Запуск захоплення та аналізу трафіку
/api/stop	POST	Зупинка захоплення та аналізу трафіку
/api/export/threats.csv	GET	Експорт журналу загроз у форматі CSV
/api/export/packets.pcap	GET	Експорт буфера перехоплених пакетів у форматі pcap

Використання WebSocket замість класичного HTTP-polling забезпечує час доставки події від моменту виявлення загрози до відображення у браузері менше однієї секунди. [26,29] При HTTP-polling браузер змушений регулярно надсилати запити на сервер щоб дізнатися про нові події, тоді як WebSocket підтримує

постійне з'єднання і сервер надсилає подію клієнту одразу у момент її виникнення без очікування запиту.

Таким чином, виходить, що багаторівнева конвеєрна архітектура забезпечує чіткий розподіл відповідальності між модулями системи. Уніфікація пакетів через структуру ParsedPacket ізолює рівень захоплення від рівня аналізу і спрощує незалежне тестування кожного детектора. Ізоляція рівня захоплення від рівня аналізу через чергу пакетів розміром 10 000 елементів гарантує стійкість до пікових навантажень без втрати даних. Фіксований порядок виклику детекторів з механізмом передачі контексту між ними усуває дублювання сповіщень і підвищує точність класифікації загроз. Використання WebSocket забезпечує час реакції менше однієї секунди від захоплення пакета до сповіщення адміністратора.

3.3. Реалізація статистичних детекторів аномалій (DDoS, DNS Tunneling)

Статистичні детектори аналізують не окремий пакет, а динаміку трафіку у часовому вікні. Клас DDoSDetector реалізує чотири незалежні методи виявлення атак, а клас DNSTunnelDetector виявляє приховані DNS-канали за трьома ознаками аномальної поведінки.

3.3.1. Детектор DDoS-атак

Детектор DDoS реалізовано у файлі ddos_detector.py. Він підтримує чотири методи детекції які працюють паралельно для кожного пакета. [13,14] Методи та умови їх спрацювання наведено у таблиці 3.10.

Таблиця 3.10 – Методи детекції DDoS-атак

Метод	Умова спрацювання	Мінімум даних
Flood	Кількість пакетів від IP за вікно перевищує DDOS_FLOOD_THRESHOLD	Будь-яка кількість
Early Flood	Більше 3000 пакетів від IP за останні 2 секунди	Будь-яка кількість
Spike	Поточна кількість пакетів	5 завершених вікон

	перевищує середнє в DDOS_THRESHOLD_MULTIPLIER разів	
Multi-Source	Більше 20 унікальних IP атакують один dst_ip за 15 секунд	500 пакетів на dst_ip

Для кожного IP-джерела система підтримує ковзне вікно пакетів у вигляді черги з двох кінців:

```
self._windows[ip].append(now)
window = self._windows[ip]
while window and window[0] < now - ws:
   window.popleft()
current = len(window)
```

Щоразу коли надходить новий пакет, його мітка часу додається в кінець черги. Одночасно з початку черги видаляються всі мітки старші за DDOS_WINDOW_SIZE секунд. Таким чином current завжди містить точну кількість пакетів від даного IP за останні 30 секунд без жодного перебору всієї структури.

Метод Spike додатково веде історію завершених вікон:

```
if now - self._window_start[ip] >= ws:
   self._history[ip].append(len(window))
   self._window_start[ip] = now
```

Коли накопичено щонайменше 5 вікон і середнє навантаження перевищує 100 пакетів за вікно, система вираховує коефіцієнт відхилення. Якщо поточне вікно перевищує середнє у 10 разів або більше,то фіксується атака типу Spike. Вимога мінімального середнього у 100 пакетів запобігає хибним спрацюванням на малоактивних мережах де навіть незначний сплеск дає великий коефіцієнт відхилення.

Механізм ігнорування IP після PortScan реалізовано через словник з мітками часу:

```
if ip in self._portscan_ips:
   if time.time() - self._portscan_ips[ip] < 30:
      return None
   else:
      del self._portscan_ips[ip]
```

Якщо IP позначений як сканер менше 30 секунд тому, детектор DDoS повністю ігнорує його трафік. Після закінчення 30 секунд запис автоматично видаляється і IP знову перевіряється у штатному режимі. Блок-схему алгоритму детекції DDoS наведено на рисунку 3.2.

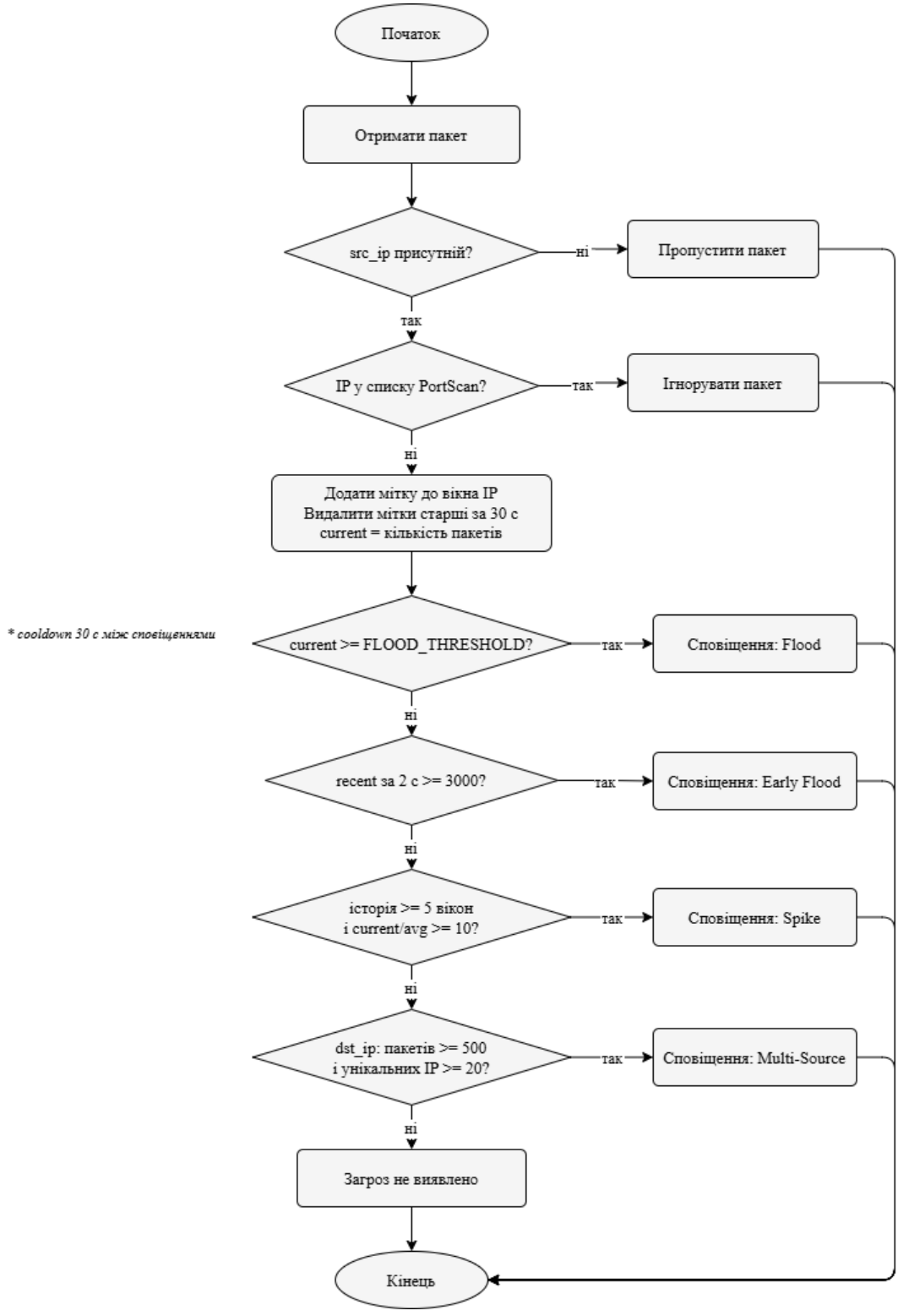

Рисунок 3.2 – Блок-схема алгоритму детектора DDoS-атак

Порогові значення детектора DDoS наведено у таблиці 3.11.

Розмір ковзного вікна DDOS_WINDOW_SIZE = 30 секунд обрано як компроміс між швидкістю реакції і стабільністю статистики. Менше вікно реагує швидше але частіше дає хибні спрацювання на легітимних сплесках трафіку, більше вікно є стійкішим але може пропустити короткочасні атаки.

Поріг Flood DDOS_FLOOD_THRESHOLD = 10 000 пакетів відповідає інтенсивності близько 333 пакетів на секунду, що є аномально високим показником для офісної мережі де звичайне навантаження рідко перевищує 50 пакетів на секунду з одного джерела.

Множник Spike DDOS_THRESHOLD_MULTIPLIER = 10 означає що поточне навантаження від IP має перевищити середньоісторичне у 10 разів. Менше значення генерує хибні спрацювання при легітимних файлових передачах, більше значення ризикує пропустити реальні атаки.

Параметр Multi-Source IP MULTI_SRC_THRESHOLD = 20 визначає мінімальну кількість унікальних IP-адрес які одночасно атакують один сервер. Значення 20 відсіває легітимні сценарії де кілька користувачів одночасно звертаються до одного ресурсу і фіксує лише справжні розподілені атаки з великої кількості джерел.

Cooldown COOLDOWN = 30 секунд запобігає лавині сповіщень при тривалій атаці. Система надсилає одне сповіщення і наступне не раніше ніж через 30 секунд навіть якщо атака продовжується.

Таблиця 3.11 – Порогові значення детектора DDoS

Параметр	Файл config.py	Значення	Опис
Розмір вікна	DDOS_WINDOW_SIZE	30 с	Тривалість ковзного вікна
Поріг Flood	DDOS_FLOOD_THRESHOLD	10 000	Абсолютна кількість пакетів
Поріг Spike	DDOS_MIN_PACKETS	2 000	Мінімум пакетів для Spike
Множник Spike	DDOS_THRESHOLD_MULTIPLIER	10	Коефіцієнт відхилення від середнього

Поріг Early Flood	—	3 000 за 2 с	Вбудований у код
Cooldown	COOLDOWN	30 с	Мінімальний інтервал між сповіщеннями
Multi-Source вікно	MULTI_SRC_WINDOW	15 с	Вікно для підрахунку джерел
Multi-Source IP	MULTI_SRC_THRESHOLD	20	Мінімум унікальних IP

3.3.2. Детектор DNS Tunneling

Детектор DNS Tunneling реалізовано у файлі dns_tunnel_detector.py. Він аналізує виключно пакети з протоколом DNS і для кожного IP-джерела веде ковзне вікно записів за останні 30 секунд. [15,16] Виявлення відбувається за трьома незалежними ознаками аномальної поведінки.

Для кожного DNS-пакета система зберігає пару (мітка часу, розмір):

```
self._records[ip].append((now, pkt.size))
records = self._records[ip]
while records and records[0][0] < now - DNS_WINDOW:
   records.popleft()
```

Після оновлення вікна обчислюються агрегатні метрики:

```
sizes = [s for _, s in records]
avg_size = sum(sizes) / count
total_bytes = sum(sizes)
large_count = sum(1 for s in sizes if s > DNS_SIZE_THRESHOLD)
```

Перша ознака тунелювання - аномально великий розмір DNS-пакетів. Легітимний DNS-запит для звичайного доменного імені рідко перевищує 100 байт. [14] Якщо більше 40 відсотків пакетів від одного IP перевищують 250 байт і загальна кількість запитів не менше 20, система фіксує аномалію з типом large_packets.

Друга ознака - аномально висока частота DNS-запитів. Якщо за 30 секунд від одного IP надійшло понад 300 DNS-запитів, це вказує на автоматизовану передачу даних через DNS-канал оскільки звичайний користувач генерує набагато менше запитів навіть при інтенсивному перегляді сайтів.

Третя ознака - великий сумарний обсяг DNS-трафіку. Якщо за 30 секунд від одного IP передано понад 30 кілобайт через DNS, система фіксує аномалію з типом high_volume оскільки такий обсяг є характерним для передачі файлів а не для звичайної роботи з доменними іменами. Блок-схему алгоритму детекції DNS Tunneling наведено на рисунку 3.3.

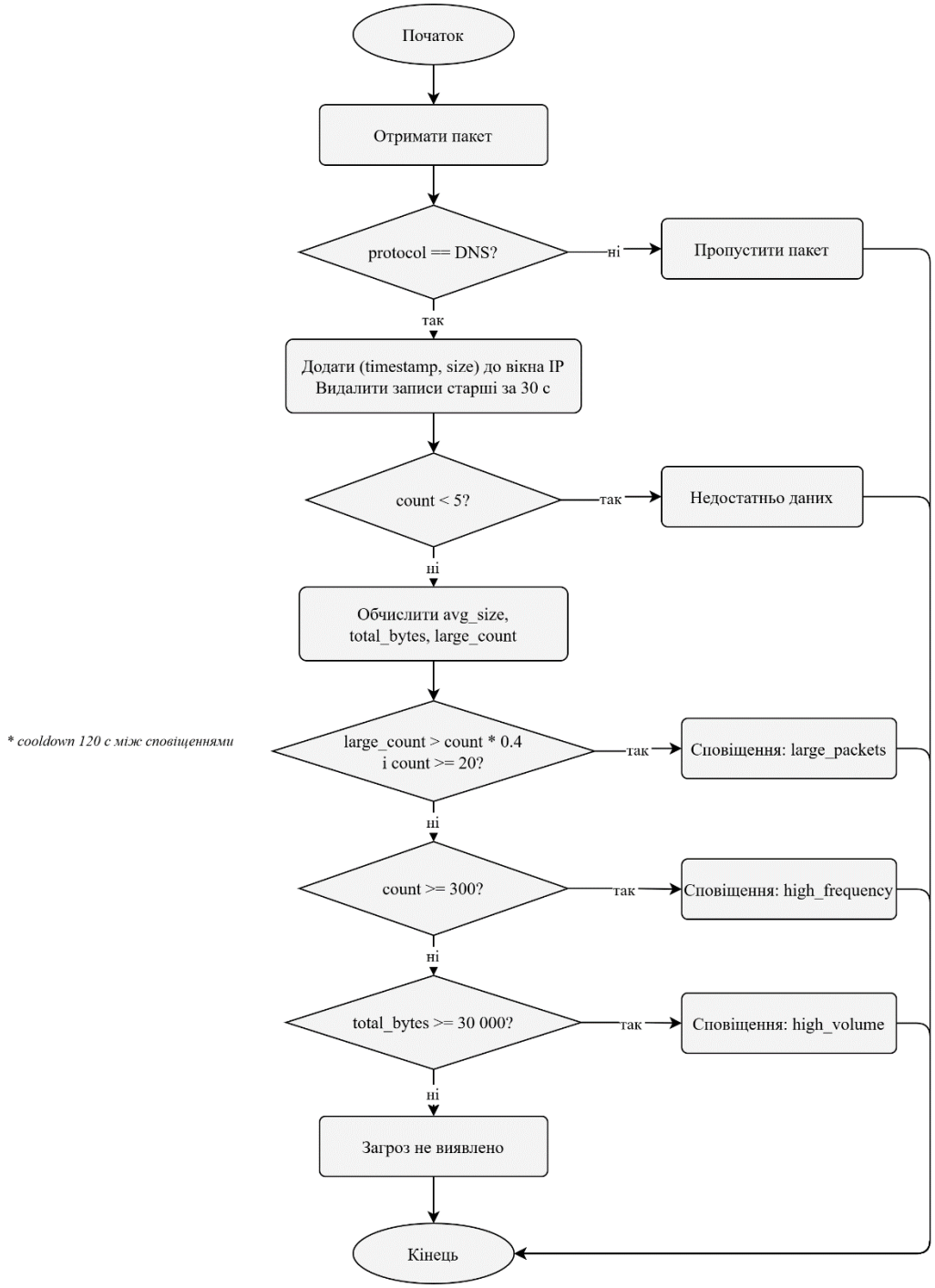

Рисунок 3.3 – Блок-схема алгоритму детектора DNS Tunneling

Порогові значення детектора DNS Tunneling наведено у таблиці 3.12.

Поріг розміру пакета DNS_SIZE_THRESHOLD = 250 байт обрано на основі того що легітимний DNS-запит рідко перевищує 100 байт. Значення 250 байт залишає достатній запас щоб не спрацьовувати на довгих але легітимних доменних іменах і при цьому надійно виявляти пакети з прихованими даними.

Поріг частоти DNS_COUNT_THRESHOLD = 300 запитів за 30 секунд відповідає інтенсивності 10 запитів на секунду. Звичайний браузер при активному перегляді сайтів генерує не більше 2-3 DNS-запитів на секунду, тому перевищення порогу у 10 разів є надійною ознакою автоматизованої передачі даних.

Поріг обсягу DNS_VOLUME_THRESHOLD = 30 000 байт за 30 секунд відповідає середньому розміру текстового файлу. Такий обсяг є нормальним для HTTP-трафіку але аномальним для DNS який призначений виключно для розпізнавання імен.

Cooldown COOLDOWN = 120 секунд для DNS Tunneling є значно довшим ніж для DDoS оскільки тунелювання є повільним і прихованим процесом, і повторне сповіщення через дві хвилини є достатнім для інформування адміністратора без надмірного навантаження журналу.

Таблиця 3.12 – Порогові значення детектора DNS Tunneling

Параметр	**Константа**	**Значення**	**Опис**
Часове вікно	DNS_WINDOW	30 с	Тривалість вікна аналізу
Поріг розміру пакета	DNS_SIZE_THRESHOLD	250 байт	Межа великого DNS-пакета
Поріг частоти	DNS_COUNT_THRESHOLD	300 запитів	Максимум запитів за вікно
Поріг обсягу	DNS_VOLUME_THRESHOLD	30 000 байт	Максимум трафіку за вікно
Частка великих пакетів	LARGE_RATIO	0.4	Мінімальна частка для спрацювання
Мінімум пакетів	MIN_PACKETS	20	Мінімум для перевірки large_packets
Cooldown	COOLDOWN	120 с	Мінімальний

			інтервал між сповіщеннями

Таким чином, реалізовані статистичні детектори використовують структуру ковзного вікна на основі deque що забезпечує ефективне оновлення даних без повного перебору. Детектор DDoS реалізує чотири незалежні методи виявлення що перекривають різні сценарії атак: від абсолютного флуду до розподілених атак з багатьох джерел. Детектор DNS Tunneling виявляє приховані канали за трьома ознаками аномальної поведінки без аналізу вмісту DNS-запитів, що не потребує розшифрування трафіку. Механізм cooldown в обох детекторах запобігає лавині дублюючих сповіщень при тривалих атаках.

3.4 Реалізація евристичних детекторів аномалій (PortScan, BruteForce, ARP Spoofing)

Евристичні детектори аналізують структурні ознаки трафіку а не його об'єм у часі. Кожен детектор реалізує кілька незалежних методів виявлення які перекривають різні сценарії однієї атаки.

3.4.1. Детектор сканування портів PortScan

Перед тим як атакувати мережу зловмисник зазвичай проводить розвідку, а саме сканує порти щоб дізнатися які сервіси запущені на цільовому хості. [2,3] Детектор PortScan реалізовано у файлі portscan_detector.py і він виявляє саме цю розвідувальну активність. [30] Тобто легітимний користувач підключається до одного-двох сервісів. Зловмисник за кілька секунд намагається підключитися до десятків різних портів. Система відстежує кількість унікальних портів до яких звертається кожна IP-адреса і реагує коли ця кількість перевищує норму. Реалізовано чотири методи виявлення наведені у таблиці 3.13.

Таблиця 3.13 – Методи детекції сканування портів та їх практичний зміст

Метод	**Що відбувається в мережі**	**Ознака для системи**
SYN Scan	Зловмисник надсилає SYN на кожен порт і не	Більше 10 унікальних портів за 5 с, більше

	завершує з'єднання. Найпоширеніший тип сканування у Nmap	половини пакетів є SYN без ACK
Connect Scan	Зловмисник повністю відкриває з'єднання і одразу закриває. Залишає сліди в логах сервера	Більше 10 портів за 5 с та більше 5 RST-відповідей від закритих портів
Fast Scan	Агресивне сканування великого діапазону портів за мінімальний час	Більше 30 унікальних портів за 5 с
Slow Scan	Навмисно уповільнене сканування, щоб не спрацювали системи захисту	Більше 30 унікальних портів за 60 с

Для кожної IP-адреси система зберігає чергу записів де кожен запис містить час пакета, номер порту і значення TCP-прапорців:

```
self._records[ip].append(
   (now, pkt.dst_port, pkt.flag_syn, pkt.flag_ack, pkt.flag_rst, pkt.flag_fin)
)
```

Зберігання всіх прапорців разом з портом в одному записі дозволяє одним проходом по черзі одночасно підрахувати кількість унікальних портів, частку SYN-пакетів і кількість RST-відповідей. Це важливо бо кожен з цих показників сам по собі може бути нормальним, але їх поєднання вказує на конкретний тип сканування.

Ефемерні порти вище 49151 система ігнорує:

```
if pkt.dst_port >= 49152:
   return None
```

Це необхідно тому що операційна система призначає клієнтським з'єднанням випадкові порти саме з цього діапазону. Без цього фільтра звичайний веб-браузер який відкриває багато вкладок міг би виглядати як сканер.

Ключова частина детектора - перевірка SYN Scan:

```
total = syn + ack
if total > 0 and (syn / total) >= SYN_RATIO_THRESHOLD:
   return self._build(ip, ports, syn, ack, rst, syn / total, "SYN Scan")
```

Якщо більше половини пакетів від IP є SYN без ACK і при цьому кількість унікальних портів перевищує поріг, то це класичне напівзакрите сканування. Зловмисник навмисно не завершує TCP-з'єднання щоб не залишати записів у логах сервера, але система все одно бачить ці спроби на рівні пакетів.

Перевірка відбувається у два етапи: спочатку аналізується швидке вікно 5 секунд, і лише якщо воно нічого не виявило,то повільне вікно 60 секунд. Це дозволяє реагувати на агресивне сканування майже миттєво і при цьому не пропускати навмисно уповільнені атаки. Порогові значення детектора наведено у таблиці 3.14.

Таблиця 3.14 – Порогові значення детектора PortScan та їх обґрунтування

Параметр	Значення	Чому саме таке значення
Вікно швидкого скану	5 с	Достатньо для виявлення Nmap за замовчуванням, який сканує ~1000 портів за секунду
Вікно повільного скану	60 с	Перекриває сценарій, де зловмисник сканує по одному порту кожні 2 секунди
Поріг швидкого скану	10 портів	Легітимний користувач рідко відкриває більше 2–3 нових з'єднань за 5 секунд
Поріг повільного скану	30 портів	За хвилину навіть активний користувач не звертається до 30 різних сервісів
Частка SYN	50%	Нижче цього значення SYN-пакети є нормальною частиною трафіку
Поріг RST	5	Кілька RST — норма. П'ять і більше за короткий час вказують на сканування закритих портів

Cooldown	10 с	Найкоротший серед усіх детекторів, бо кожна нова хвиля сканування потребує окремого сповіщення

Блок-схему алгоритму роботи детектора PortScan наведено на рисунку 3.4.

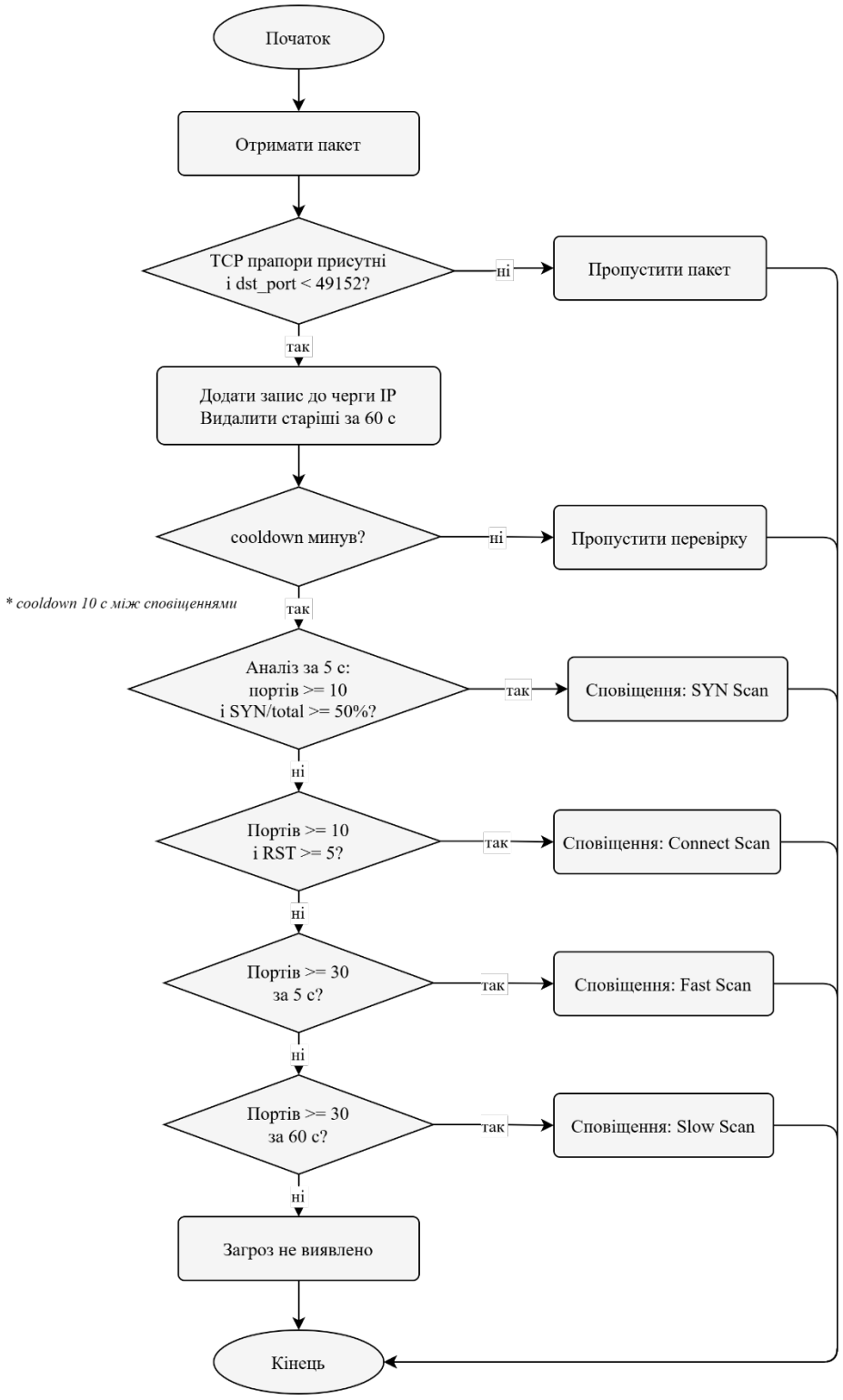

Рисунок 3.4 – Блок-схема алгоритму детектора PortScan

3.4.2. Детектор підбору паролів BruteForce

Атака BruteForce полягає в автоматизованому переборі паролів до сервісів які приймають підключення по мережі. [17,9] Детектор реалізовано у файлі bruteforce_detector.py і він відстежує спроби підключення на одинадцять сервісних портів які є найпоширенішими цілями таких атак. Перелік портів наведено у таблиці 3.15.

Таблиця 3.15 – Сервісні порти що відстежуються детектором BruteForce

Порт	Сервіс	Чому є ціллю атак
22	SSH	Основний протокол віддаленого управління Linux-серверами
3389	RDP	Віддалений робочий стіл Windows, часто відкритий назовні
21	FTP	Передача файлів, часто зі слабкими паролями
23	TELNET	Застарілий протокол без шифрування
25	SMTP	Поштовий сервер, ціль для спам-розсилок
445	SMB	Мережеві папки Windows, вразливі до багатьох атак
3306	MySQL	База даних, часто доступна без пароля за замовчуванням
5432	PostgreSQL	База даних
110	POP3	Пошта, часто зі слабкими паролями
143	IMAP	Пошта
8080	HTTP-ALT	Альтернативний веб-порт, часто адмін-панелі

Детектор реагує тільки на SYN-пакети без ACK або RST-пакети що надходять на ці порти:

```
if not pkt.flag_syn and not pkt.flag_rst:
    return None
if pkt.flag_syn and pkt.flag_ack:
    return None
```

Перша умова відкидає пакети які не є спробою підключення. Друга відкидає SYN-ACK пакети, це відповідь сервера на легітимний запит клієнта, тобто нормальна мережева активність а не атака. Завдяки цьому фільтру детектор не реагує на трафік вже встановлених сесій. Реалізовані три методи виявлення наведені у таблиці 3.16.

Таблиця 3.16 – Методи детекції BruteForce та їх практичний зміст

Метод	Що відбувається в мережі	Умова спрацювання
Fast	Програма для підбору паролів працює на максимальній швидкості. За 30 секунд встигає спробувати сотні паролів	Більше 15 спроб на один порт за 30 с з однієї IP
Slow	Зловмисник навмисно уповільнює підбір, щоб не перевищувати пороги швидких систем захисту	Більше 40 спроб на один порт за 2 хвилини з однієї IP

Для виявлення розподіленої атаки система веде два незалежних трекери одночасно:

```
self._records[ip].append((now, port))
self._dist_records[port].append((now, ip))
```

Перший трекер індексований по IP-адресі і дозволяє виявляти атаки з одного джерела. Другий трекер індексований по порту і дозволяє побачити що до одного сервісу підключаються багато різних IP одночасно. Без другого трекера розподілена атака була б невидима бо кожна окрема IP-адреса робить мало спроб і не перевищує індивідуальних порогів.

Перевірка розподіленої атаки:

```
unique_ips = len(set(s for _, s in dist))
if unique_ips >= DIST_THRESHOLD and len(dist) >= DIST_PKTS:
   return BruteForceThreat(f"{unique_ips} IPs", now, len(dist), port, service, "Distributed")
```

Система підраховує кількість унікальних IP-адрес які зверталися до одного порту за останні 30 секунд. Якщо їх більше п'яти і загальна кількість пакетів перевищує 20, то фіксується координована атака. Порогові значення детектора наведено у таблиці 3.17.

Таблиця 3.17 – Порогові значення детектора BruteForce та їх обґрунтування

Параметр	Значення	Чому саме таке значення
Поріг Fast	15 спроб за 30 с	Людина, яка помиляється з паролем, робить максимум 5–6 спроб і зупиняється. 15 спроб за 30 с це вже точно автоматизований підбір
Поріг Slow	40 спроб за 120 с	Дозволяє виявити атаку, яка навмисно розтягнута на дві хвилини, щоб обійти Fast-поріг
Поріг Distributed IP	5 IP	Менше 5 IP могло бути випадковим збігом. 5 і більше вказують на координацію
Поріг Distributed пакетів	20 пакетів	Мінімальна кількість, щоб відрізнити реальну атаку від випадкового збігу
Cooldown	60 с	Достатньо, щоб адміністратор отримав сповіщення і не отримував дублікати кожні кілька секунд

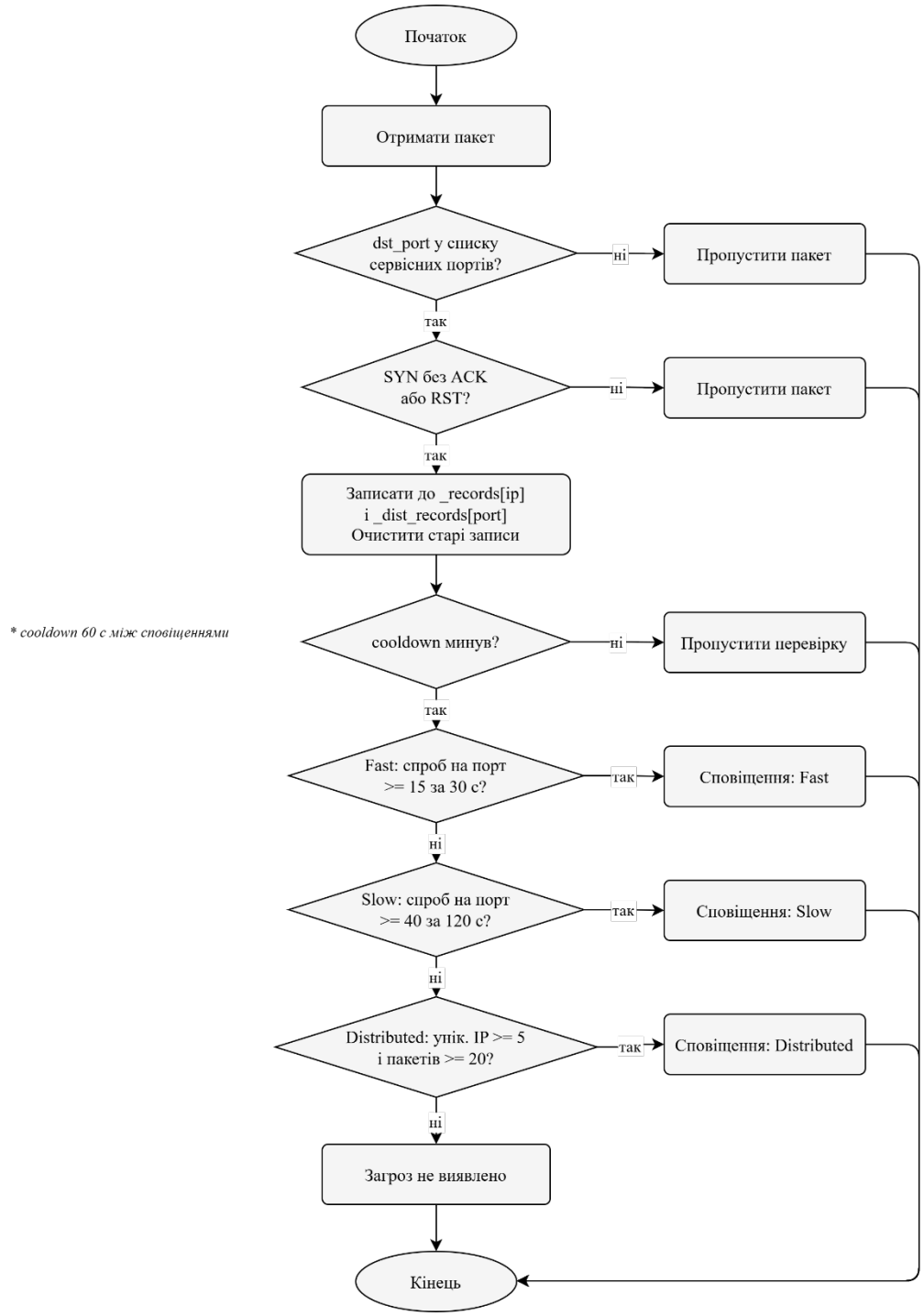

Рисунок 3.5 – Блок-схема алгоритму детектора BruteForce

3.4.3. Детектор підміни ARP-адрес

ARP Spoofing – атака де зловмисник надсилає підроблені ARP-повідомлення у локальну мережу щоб прив'язати свою MAC-адресу до IP-адреси іншого пристрою. [21] В результаті трафік призначений для жертви починає надходити до зловмисника. Якщо підміняється адреса шлюзу, то зловмисник отримує весь трафік мережі що виходить в інтернет.

Детектор реалізовано у файлі arp_spoof_detector.py. На відміну від інших детекторів він отримує сирі пакети безпосередньо від сніфера ще до їх розбору у структуру ParsedPacket. Це необхідно тому що MAC-адреса зберігається в ARP-заголовку і не передається далі по конвеєру. Реалізовано три методи виявлення наведені у таблиці 3.18.

Таблиця 3.18 – Методи детекції ARP Spoofing та їх практичний зміст

Метод	Що відбувається в мережі	Як система це виявляє
MAC Change	Пристрій з IP-адресою 192.168.1.5 раптово починає представлятися іншою MAC-адресою. Це головна ознака ARP Spoofing	Нова MAC-адреса для IP, яка вже є у таблиці, не збігається з раніше записаною
Gateway Spoof	Зловмисник видає себе за шлюз мережі, щоб перехоплювати весь трафік. Найнебезпечніший сценарій	Підміна MAC виявлена саме для IP-адреси шлюзу — класифікується окремо через критичну небезпеку
ARP Flood	Зловмисник масово розсилає підроблені ARP-відповіді, щоб отруїти кеші всіх пристроїв мережі одночасно	Більше 10 ARP-відповідей за 10 секунд від одного IP

Основна логіка детектора побудована на таблиці відповідності IP і MAC-адрес:

```
if src_ip in self._ip_mac_table:
    known_mac = self._ip_mac_table[src_ip]
    if known_mac != src_mac:
        attack_type = "Gateway Spoof" if src_ip == self._gateway_ip else "MAC Change"
        return ARPSpoofThreat(src_ip, now, src_ip, src_mac, original, attack_type)
else:
    self._ip_mac_table[src_ip] = src_mac
```

Коли система вперше бачить IP-адресу вона запам'ятовує її MAC як легітимний. При кожному наступному ARP-пакеті від цієї адреси вона порівнює поточну MAC з тією що записана. Якщо MAC змінилась - атака. Якщо це перша

поява IP у мережі, то просто записуємо і продовжуємо. Такий підхід не потребує жодних попередніх налаштувань і автоматично навчається у процесі роботи.

Адреса шлюзу визначається автоматично:

```
if src_ip.endswith('.1') or src_ip.endswith('.254'):
    self._gateway_ip = src_ip
```

У більшості мереж шлюз має адресу що закінчується на .1 або .254. Якщо підміна стосується саме цієї адреси, то атака класифікується як Gateway Spoof що виводиться у журналі загроз з окремим позначенням через критичну небезпеку.

Детектор аналізує виключно ARP-відповіді з кодом операції op=2 і ігнорує ARP-запити з кодом op=1. [21] Це принципово важливо: ARP-запит лише питає "у кого є ця IP-адреса" і сам по собі нешкідливий. ARP-відповідь стверджує "ця IP належить мені" і саме через підроблені відповіді відбувається отруєння кешу.Порогові значення детектора наведено у таблиці 3.19.

Таблиця 3.19 – Порогові значення детектора ARP Spoofing та їх обґрунтування

Параметр	Значення	Чому саме таке значення
Вікно ARP Flood	10 с	Коротке вікно, бо масове отруєння кешу відбувається дуже швидко
Поріг ARP Flood	10 відповідей за 10 с	У нормальній мережі вузол надсилає ARP-відповіді тільки у відповідь на запити інших — їх дуже мало. Тому 1 відповідь на секунду без запитів є явно ознакою атаки
Cooldown	30 с	Баланс між своєчасністю сповіщень і відсутністю дублікатів

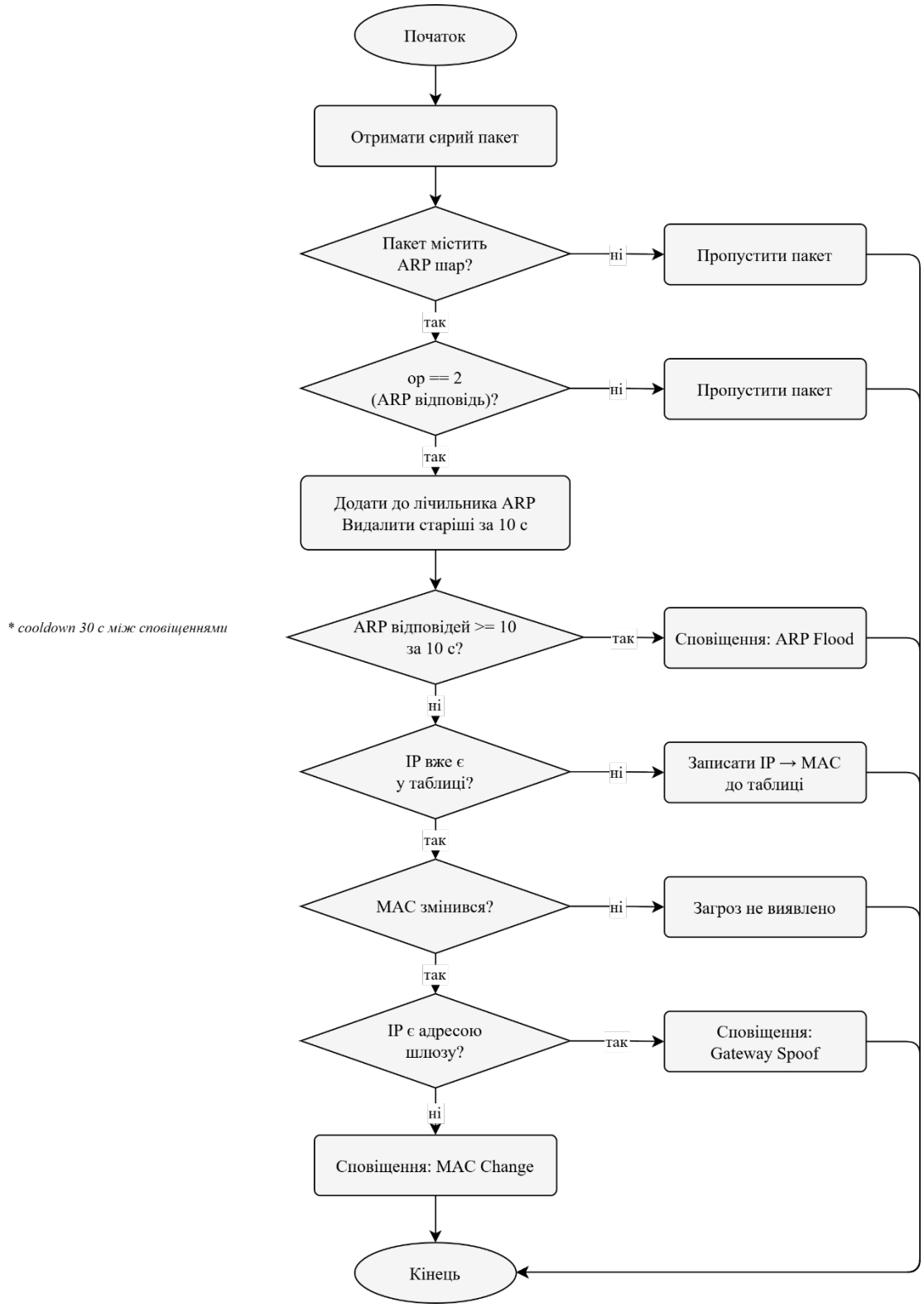

Рисунок 3.6 – Блок-схема алгоритму детектора ARP Spoofing

Реалізовані евристичні детектори виявляють загрози за структурними ознаками поведінки а не за об'ємом трафіку. Детектор PortScan розрізняє чотири типи сканування через аналіз унікальних портів і TCP-прапорців у двох часових вікнах. Детектор BruteForce відстежує спроби підключення на одинадцять

сервісних портів і завдяки паралельним трекерам по IP і по порту виявляє як атаки з одного джерела так і координовані атаки з кількох джерел одночасно. Детектор ARP Spoofing автоматично будує таблицю легітимних IP і MAC-адрес і реагує на будь-яку зміну включаючи найнебезпечніший сценарій підміни адреси мережевого шлюзу.

3.5 Реалізація модуля моніторингу QoS та веб-інтерфейсу

3.5.1. Модуль моніторингу якості обслуговування мережі

Модуль QoS Monitor реалізовано у файлі qos_monitor.py. Його завдання відрізняється від детекторів загроз, він не шукає атаки, а постійно вимірює три показники якості мережі: пропускну здатність, затримку і відсоток втрачених пакетів. Всі вимірювання виконуються пасивно, тобто система аналізує трафік який вже проходить через мережу і не генерує жодного додаткового пакета. Що означає кожен показник і як він вимірюється наведено у таблиці 3.20.

Кожну секунду модуль повертає об'єкт QoSMetrics з актуальними показниками:

```
@dataclass
class QoSMetrics:
    timestamp: float
    bandwidth_mbps: float
    rtt_ms: float
    packet_loss_pct: float
    total_packets: int
    total_bytes: int
    protocol_counts: dict
```

Таблиця 3.20 – Показники QoS та методи їх вимірювання

Показник	Поле	Як вимірюється	Що показує на практиці
Пропускна здатність	bandwidth_mbps	Сума байтів за інтервал, перерахована у Мбіт/с	Скільки даних реально передається через мережу зараз
Затримка RTT	rtt_ms	Різниця часу між SYN і SYN-ACK у мілісекундах	Час від відправки запиту до отримання

			відповіді сервера
Втрати пакетів	packet_loss_pct	Відношення SYN без відповіді до загальної кількості SYN	Відсоток з'єднань, які не отримали відповідь за 5 секунд
Розподіл протоколів	protocol_counts	Лічильник пакетів по кожному протоколу	Яким трафіком зайнята мережа: HTTP, SSH, RDP тощо

Пропускна здатність обчислюється як сума всіх байтів за інтервал перерахована у мегабіти за секунду:

bw = (self._bytes * 8) / (elapsed * 1_000_000)

Множення на 8 переводить байти у біти, ділення на мільйон переводить біти у мегабіти. Змінна elapsed містить тривалість поточного інтервалу у секундах яка зазвичай становить одну секунду відповідно до параметра METRICS_PUSH_INTERVAL у файлі config.py.

Найцікавіша частина модуля є вимірювання RTT без генерації тестових пакетів. Система перехоплює SYN-пакети клієнтів і SYN-ACK відповіді серверів та вимірює час між ними: [4,6]

```
def _tcp_rtt(self, pkt, now):
    if pkt.flag_syn and not pkt.flag_ack and pkt.tcp_seq is not None:
        self._pending_syn[pkt.tcp_seq] = now
        self._syn_count += 1
    elif pkt.flag_syn and pkt.flag_ack and pkt.tcp_ack_num is not None:
        t = self._pending_syn.pop(pkt.tcp_ack_num - 1, None)
        if t and 0 < now - t < 5:
            self._rtt.append(now - t)
            self._answered += 1
```

Коли система бачить SYN-пакет вона записує його порядковий номер і мітку часу у словник _pending_syn. Коли приходить SYN-ACK відповідь система знаходить відповідний запис за номером підтвердження і обчислює різницю часу. Збіг між SYN і SYN-ACK відбувається через поле tcp_ack_num - 1 яке за стандартом TCP завжди дорівнює порядковому номеру початкового SYN-пакета.

Якщо SYN-ACK не прийшов протягом 5 секунд — пакет вважається втраченим і враховується у відсотку втрат.

Відсоток втрачених пакетів обчислюється як різниця між кількістю відправлених SYN і отриманих SYN-ACK відповідей:

loss = round(max(0, (self._syn_count - self._answered) / self._syn_count) * 100, 1)

Функція max(0, ...) захищає від від'ємних значень які теоретично можливі якщо SYN-ACK надійшов до того як система встигла зафіксувати відповідний SYN. Це може статися при старті системи коли деякі з'єднання вже були встановлені до початку захоплення.

Після кожного виклику get_metrics() всі лічильники скидаються через метод _reset(). Завдяки цьому кожне значення відображає стан мережі саме за останній інтервал а не накопичені дані за весь час роботи системи.

3.5.2. Веб-інтерфейс та візуалізація даних у реальному часі

Веб-інтерфейс системи TrafficAnalyzer реалізовано у вигляді односторінкового застосунку на основі HTML, CSS і JavaScript. Уся логіка відображення зосереджена у файлі dashboard.js який містить близько 400 рядків коду. Для побудови живих графіків використано бібліотеку Chart.js, для отримання даних у реальному часі використан протокол WebSocket через бібліотеку Socket.IO. [23,24] Загальний вигляд інтерфейсу наведено на рисунку 3.7.

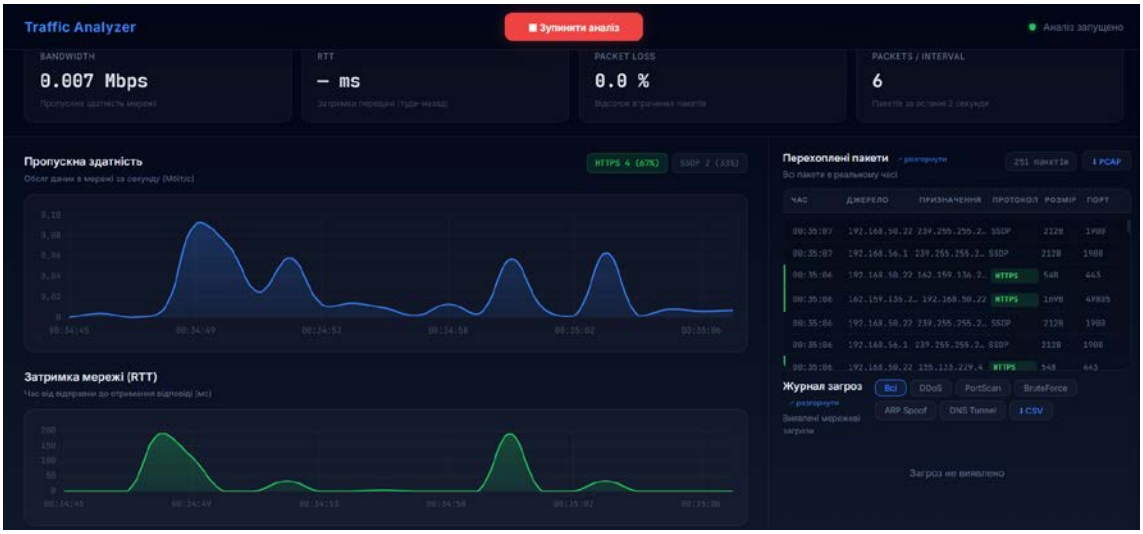

Рисунок 3.7 – Головна сторінка веб-інтерфейсу TrafficAnalyzer

3.5.2.1 Підключення до сервера та отримання даних

Весь обмін даними між браузером і сервером відбувається через WebSocket. Це означає що сервер сам надсилає дані браузеру у момент їх появи без жодних запитів з боку клієнта. [26,29] Підключення і обробка подій реалізовані так:

```
const socket = io();
socket.on('connect', () => setStatus(true));
socket.on('disconnect', () => setStatus(false));
socket.on('metrics', d => {
   elBw.textContent = d.bandwidth_mbps.toFixed(3) + ' Mbps';
   elRtt.textContent = d.rtt_ms > 0 ? d.rtt_ms.toFixed(1) + ' ms' : '— ms';
   elLoss.textContent = d.packet_loss_pct.toFixed(1) + ' %';
   pushPt(bwChart, d.bandwidth_mbps, d.timestamp);
   pushPt(rttChart, d.rtt_ms, d.timestamp);
   updateLegend(d.protocol_counts || {});
});
socket.on('threat', e => { allThreats.unshift(e); renderThreats(); });
socket.on('packet', p => { addMiniRow(p); storedPackets.push(p); });
```

Система підписується на три типи подій. Подія metrics надходить кожну секунду і містить актуальні показники QoS. Подія threat надходить миттєво у момент виявлення загрози і одразу додає картку до журналу загроз. Подія packet надходить для кожного перехопленого пакета і додає рядок до таблиці пакетів. Завдяки WebSocket затримка між виявленням загрози на сервері і появою сповіщення у браузері становить менше однієї секунди.

3.5.2.2 Панель метрик

У верхній частині інтерфейсу розміщено чотири картки з ключовими показниками мережі у реальному часі наведені на рисунку 3.8.

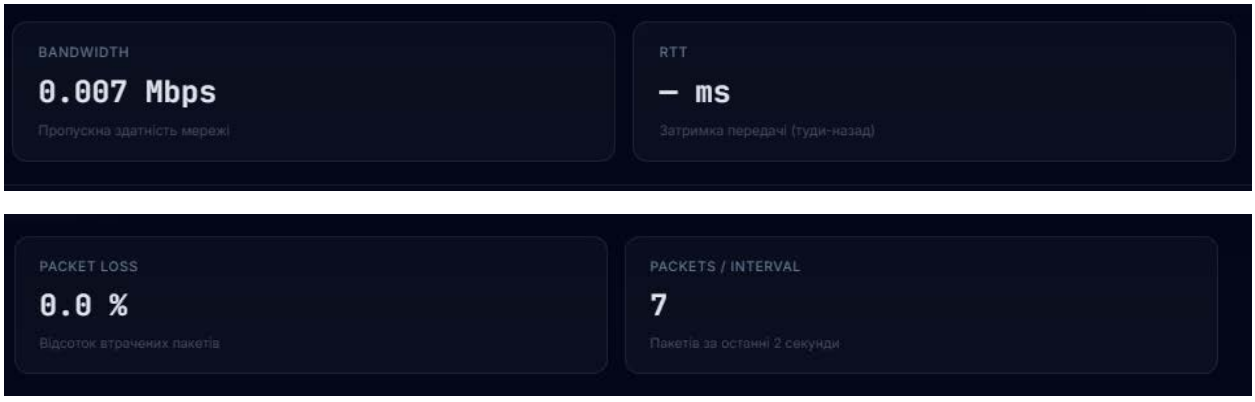

Рисунок 3.8 – Панель метрик якості мережі

Склад показників панелі метрик наведено у таблиці 3.21.

Таблиця 3.21 – Показники панелі метрик веб-інтерфейсу

Показник	Що відображає	Одиниця
Bandwidth	Поточна пропускна здатність мережі	Мбіт/с
RTT	Затримка передачі туди і назад	мс
Packet Loss	Відсоток втрачених пакетів	%

3.5.2.3 Графіки в реальному часі

Центральне місце інтерфейсу займають два живих графіки наведені на рисунку 3.9. Вони побудовані на основі бібліотеки Chart.js і оновлюються автоматично кожну секунду без перезавантаження сторінки.

Рисунок 3.9 – Графіки пропускної здатності та затримки мережі

- Верхній графік показує пропускну здатність мережі у Мбіт/с. По горизонтальній осі відкладається час у форматі годин, хвилин і секунд. По вертикальній осі відкладається обсяг трафіку. Кожна точка графіка відповідає одному секундному інтервалу вимірювання. На графіку добре видно сплески трафіку, наприклад завантаження файлу або відкриття важкої

сторінки виглядає як різкий пік після якого значення повертається до базової лінії.

- Нижній графік показує затримку мережі RTT у мілісекундах. Це час від моменту відправки SYN-пакета клієнтом до отримання SYN-ACK відповіді від сервера. У нормальному стані мережі цей показник є стабільним і близьким до нуля для локальних ресурсів або в межах 10-50 мс для зовнішніх серверів. Різкий стрибок RTT може вказувати на перевантаження каналу або на початок DDoS-атаки яка збільшує час обробки запитів.

Ініціалізація графіка пропускної здатності виконується при завантаженні сторінки:

```
const bwCtx = $('chart-bandwidth').getContext('2d');
const bwGrad = bwCtx.createLinearGradient(0, 0, 0, 210);
bwGrad.addColorStop(0, 'rgba(59,130,246,0.3)');
bwGrad.addColorStop(1, 'rgba(59,130,246,0.0)');

const bwChart = new Chart(bwCtx, {
  type: 'line',
  data: { labels: [], datasets: [{ data: [],
    borderColor: '#3b82f6', backgroundColor: bwGrad,
    borderWidth: 2, pointRadius: 0, fill: true, tension: .4,
  }]},
  options: { responsive: true, maintainAspectRatio: false,
    animation: { duration: 400 },
    scales: {
      x: { ticks: { maxTicksLimit: 6 }, grid: { color: 'rgba(255,255,255,0.04)' }},
      y: { beginAtZero: true }
    }
  }
});
```

Параметр pointRadius: 0 прибирає точки на графіку щоб лінія виглядала плавною при частих оновленнях. Параметр tension: 0.4 задає ступінь згладжування кривої між точками. Параметр fill: true разом з градієнтом bwGrad створює заливку під кривою яка візуально підкреслює площу під кривою і полегшує сприйняття динаміки навантаження. Параметр animation: { duration: 400 } задає час анімації початкової побудови графіка.

Обидва графіки мають ковзне вікно у 60 точок що відповідає рівно одній хвилині спостереження:

```
const MAX_POINTS = 60;
function pushPt(ch, v, ts) {
   ch.data.labels.push(
      new Date(ts * 1000).toLocaleTimeString('uk', { hour12: false })
   );
   ch.data.datasets[0].data.push(v);
   if (ch.data.labels.length > MAX_POINTS) {
      ch.data.labels.shift();
      ch.data.datasets[0].data.shift();
   }
   ch.update('none');
}
```

Кожну секунду функція pushPt() додає нову точку в кінець масиву і одночасно видаляє найстарішу з початку. Таким чином графік завжди показує саме останню хвилину роботи мережі і не накопичує дані за весь час роботи системи. Параметр 'none' вимикає анімацію переходу між оновленнями що забезпечує плавне відображення без мерехтіння при частих оновленнях кожну секунду.

У правому верхньому куті графіка пропускної здатності розміщено динамічну легенду протоколів. Вона показує які протоколи зараз генерують трафік, скільки пакетів кожного типу перехоплено за останній інтервал і яку частку від загального трафіку вони займають. Наприклад напис HTTPS 7 (64%) означає що за останню секунду з 11 перехоплених пакетів 7 є HTTPS трафіком що становить 64 відсотки від загального обсягу. Легенда оновлюється разом з кожним надходженням метрик і сортується від найбільшого протоколу до найменшого.

3.5.2.4 Таблиця перехоплених пакетів

У правій частині головного екрану розміщено таблицю перехоплених пакетів наведену на рисунку 3.10.

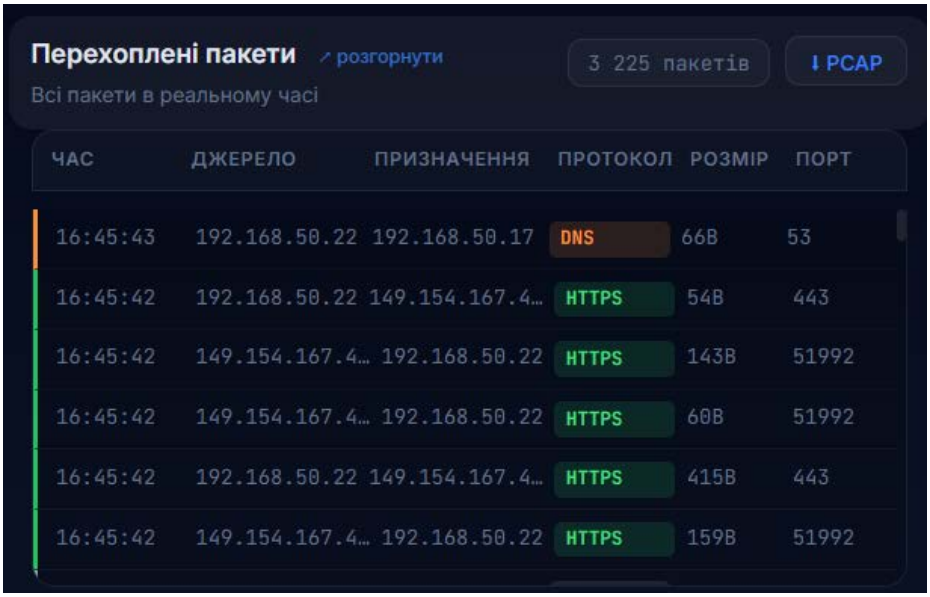

Рисунок 3.10 – Таблиця перехоплених пакетів на головному екрані

Таблиця відображає останні 200 пакетів у реальному часі з такими полями: час, IP джерела, IP призначення, протокол, розмір і порт. Кожен протокол підсвічується окремим кольором що дозволяє миттєво візуально відрізнити типи трафіку. При натисканні на заголовок таблиці відкривається розгорнута панель з повним переліком пакетів і детальною інформацією наведена на рисунку 3.11.

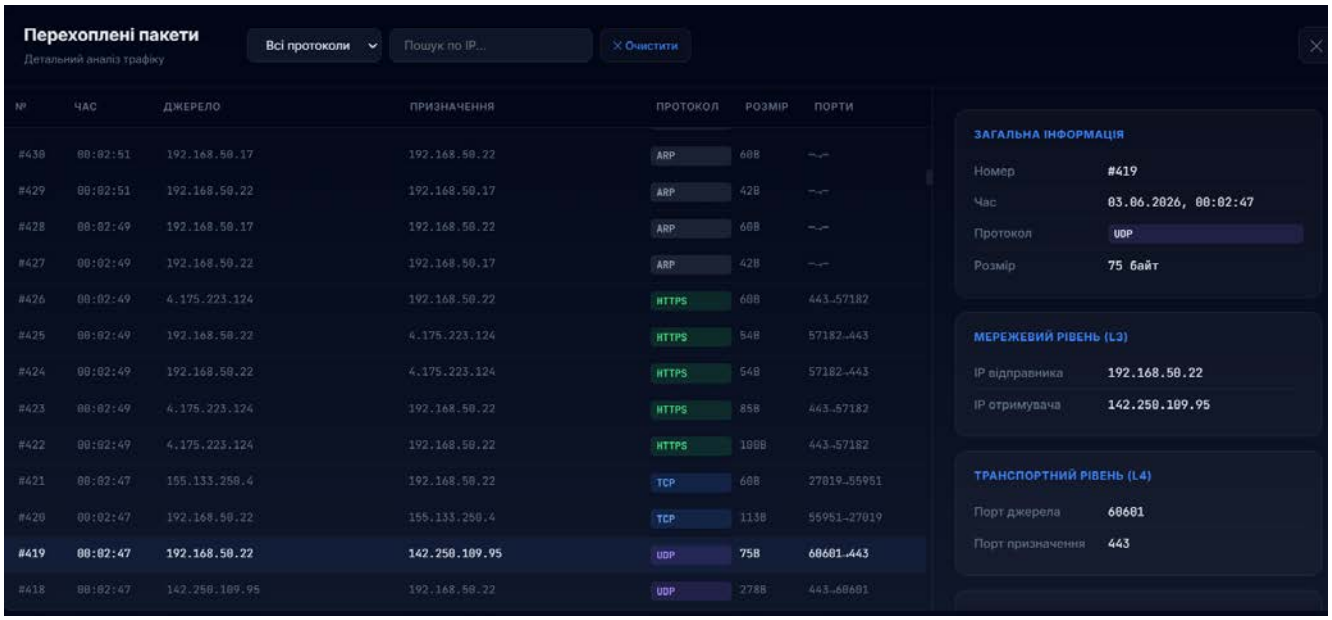

Рисунок 3.11 – Розгорнута панель перехоплених пакетів з деталями пакета

Розгорнута панель дозволяє фільтрувати пакети за протоколом і за IP-адресою. При виборі конкретного пакета у правій частині панелі відображаються

деталі: час захоплення, протокол, розмір, IP-адреси джерела і призначення, порти і опис протоколу. Система зберігає у пам'яті браузера до 2000 останніх пакетів:

```
storedPackets.push(p);
while (storedPackets.length > MAX_STORED) storedPackets.shift();
```

Це дозволяє переглядати і фільтрувати пакети навіть якщо вони вже зникли з основної таблиці. Кнопка PCAP у правому верхньому куті дозволяє завантажити буфер перехоплених пакетів у форматі pcap для подальшого аналізу у Wireshark.

3.5.2.6 Журнал загроз

Нижче таблиці пакетів розміщено журнал загроз наведений на рисунку 3.12. Кожна виявлена загроза відображається у вигляді картки з типом загрози, часом виявлення, IP-адресою джерела і короткими деталями.

Рисунок 3.12 – Журнал загроз з виявленою атакою PortScan

Картки загроз підсвічуються різними кольорами залежно від типу загрози. Над журналом розміщено кнопки фільтрації за типом: Всі, DDoS, PortScan, BruteForce, ARP Spoof, DNS Tunnel. При натисканні на заголовок журналу відкривається розгорнута панель наведена на рисунку 3.13.

Рисунок 3.13 – Розгорнута панель журналу загроз з деталями атаки PortScan

Розгорнута панель показує повний журнал з можливістю фільтрації за типом загрози і за IP-адресою. При виборі конкретної загрози у правій частині відображаються повні деталі включаючи тип атаки, час виявлення, IP джерело, технічні параметри і пояснення що означає ця загроза людською мовою. Наприклад для PortScan відображається список просканованих портів, коефіцієнт SYN і тип сканування. Для DDoS відображається кількість пакетів у вікні, середнє значення і коефіцієнт відхилення.

Рендеринг картки загрози реалізовано через функцію threatCard():

```
function threatCard(e) {
  const c = document.createElement('div');
  c.className = `threat-card ${e.type.toLowerCase().replace(/\s+/g, '-')}`;
  const t = new Date(e.timestamp * 1000).toLocaleTimeString('uk', { hour12: false });
  let d = '';
  if (e.type === 'PortScan') {
    d = `${e.unique_ports} портів`;
    if (e.scan_type) d += ` [${e.scan_type}]`;
    d += `, SYN ratio ${e.syn_ratio}`;
    if (e.scanned_ports?.length) d += ` · [${e.scanned_ports.join(', ')}]`;
  }
  c.innerHTML = `
    <div class="threat-card-header">
      <span class="threat-type">${e.type}</span>
      <span class="threat-time">${t}</span>
    </div>
    <div class="threat-ip">${e.src_ip}</div>
    <div class="threat-detail">${d}</div>`;
  return c;
```

```
}
```

Функція динамічно формує HTML картки залежно від типу загрози. CSS клас призначається автоматично на основі типу загрози що забезпечує відповідний колір підсвічування без додаткової логіки.

3.5.2.6 Керування системою та експорт даних

Кнопка запуску і зупинки аналізу розміщена по центру верхньої панелі. При натисканні вона надсилає POST запит на відповідний endpoint і змінює свій вигляд залежно від стану системи:

```
function toggleAnalyzer() {
   fetch(isRunning ? '/api/stop' : '/api/start', { method: 'POST' })
      .then(r => r.json())
      .then(d => {
         isRunning = d.status === 'started';
         updateBtn();
         if (isRunning) clearAll();
      });
}
```

При запуску аналізу функція clearAll() очищує всі графіки, таблиці і журнал загроз щоб нова сесія моніторингу починалася з чистого стану. Система підтримує два формати експорту даних: журнал загроз у форматі CSV через кнопку у журналі загроз і буфер перехоплених пакетів у форматі pcap через кнопку у таблиці пакетів. Обидва файли завантажуються безпосередньо у браузері через відповідні REST endpoint.

З цього виходить, що веб-інтерфейс TrafficAnalyzer забезпечує відображення стану мережі у реальному часі без необхідності оновлення сторінки. Використання WebSocket замість HTTP-polling дозволяє отримувати сповіщення про загрози з затримкою менше однієї секунди. Живі графіки Chart.js з ковзним вікном 60 точок дають наочне уявлення про динаміку пропускної здатності і затримки мережі. Розгорнуті панелі пакетів і загроз з фільтрацією і детальним поясненням кожної події забезпечують адміністратору повний контекст для прийняття рішень.

Висновок до розділу 3

1. У третьому розділі розроблено програмний комплекс TrafficAnalyzer який реалізує метод автоматизованого аналізу мережевого трафіку в реальному часі. Розробка охоплює повний стек від рівня захоплення пакетів до веб-інтерфейсу адміністратора.
2. Обґрунтовано вибір технологічного стеку. Бібліотека Scapy обрана як єдиний інструмент серед розглянутих альтернатив який забезпечує нативний розбір пакетів на рівнях L2–L7 без залежності від зовнішніх процесів. [19] Flask з розширенням Flask-SocketIO обрано через нативну підтримку WebSocket у threading-режимі що забезпечує час доставки події менше однієї секунди. [25,26] Chart.js обрано як єдиний інструмент візуалізації який не потребує окремої серверної інфраструктури. [27]
3. Реалізовано шестирівневий конвеєр обробки трафіку де кожен рівень виконує єдину чітко визначену функцію. Уніфікація пакетів через структуру ParsedPacket ізолює рівень захоплення від рівня аналізу і забезпечує незалежне тестування кожного детектора. Черга пакетів розміром 10 000 елементів гарантує стійкість системи до пікових навантажень без втрати даних.
4. Реалізовано п'ять незалежних детекторів загроз які охоплюють основні класи мережевих атак. Детектор DDoS реалізує чотири методи виявлення включаючи Flood, Early Flood, Spike і Multi-Source що перекривають різні сценарії атак на відмову в обслуговуванні. Детектор DNS Tunneling виявляє приховані канали за трьома ознаками аномальної поведінки без аналізу вмісту запитів. Детектор PortScan розрізняє чотири типи сканування через аналіз унікальних портів і TCP-прапорців у двох часових вікнах. Детектор BruteForce виявляє як швидкі так і навмисно уповільнені атаки а також координовані атаки з кількох джерел одночасно завдяки паралельним трекерам по IP і по порту. Детектор ARP Spoofing автоматично будує

таблицю легітимних адрес і виявляє підміну включаючи найнебезпечніший сценарій підміни адреси шлюзу.

5. Реалізовано модуль пасивного вимірювання показників якості мережі який розраховує пропускну здатність, затримку RTT і відсоток втрачених пакетів без генерації тестового трафіку. Веб-інтерфейс забезпечує відображення всіх показників у реальному часі через WebSocket з затримкою менше однієї секунди від моменту виявлення події до її відображення у браузері.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Методика та середовище тестування

Тестування програмного комплексу TrafficAnalyzer проводилось у контрольованому середовищі що складається з двох вузлів: хостової машини де запускалась система моніторингу і віртуальної машини з якої генерувались тестові атаки.

4.1.1 Середовище тестування

Хостова машина з запущеним TrafficAnalyzer має характеристики наведені у таблиці 4.1.

Таблиця 4.1 – Характеристики хостової машини

Параметр	Значення
Операційна система	Windows 10
Процесор	Intel Core i7-11370H, 3.3 ГГц, 4 ядра, 8 потоків
Оперативна пам'ять	16 ГБ
Мережевий інтерфейс	Ethernet
IP-адреса основного інтерфейсу	192.168.50.22
IP-адреса Host-only інтерфейсу	192.168.56.1

Для генерації тестових атак використано віртуальну машину Ubuntu Server 24 розгорнуту у VirtualBox з характеристиками наведеними у таблиці 4.2. [31,32]

Таблиця 4.2 – Характеристики віртуальної машини для генерації атак

Параметр	Значення
Операційна система	Ubuntu Server 24 (64-bit)
Процесор	2 vCPU
Оперативна пам'ять	2048 МБ
Мережевий адаптер 1	Host-only, IP: 192.168.56.101
Мережевий адаптер 2	NAT (для доступу до інтернету)

Хостова машина має два активних мережевих інтерфейси: основний Ethernet з адресою 192.168.50.22 через який підключена до робочої мережі, і віртуальний Host-only інтерфейс VirtualBox з адресою 192.168.56.1 який

використовується виключно для зв'язку з віртуальними машинами. TrafficAnalyzer запускався на хостовій машині і прослуховував Host-only інтерфейс. Віртуальна машина Ubuntu Server з адресою 192.168.56.101 підключена до того ж Host-only сегменту через перший мережевий адаптер і виступала джерелом тестових атак. Другий NAT-адаптер віртуальної машини використовувався лише для доступу до інтернету і не брав участі у тестуванні. Такий підхід дозволяє повністю ізолювати тестовий трафік від робочої мережі і при цьому тестувати систему на реальних мережевих пакетах. Схему тестового середовища наведено на рисунку 4.1.

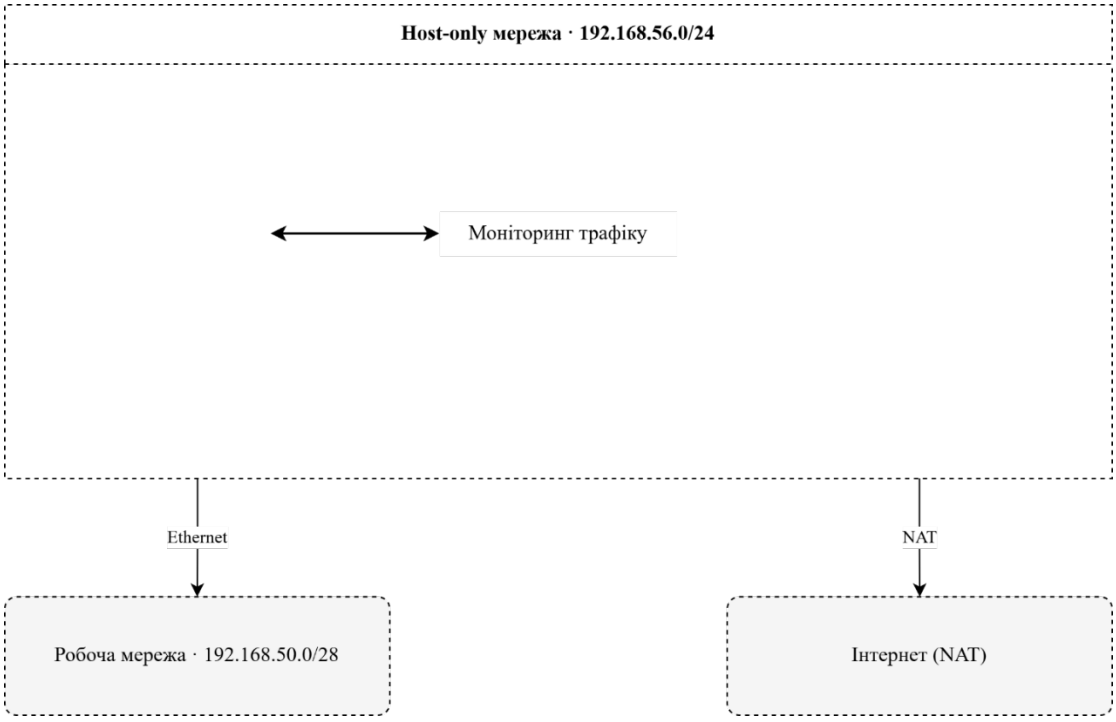

Рисунок 4.1 – Схема тестового середовища

4.1.2 Інструменти генерації атак

Для тестування кожного детектора використано окремий інструмент що максимально точно відтворює реальну атаку. Перелік інструментів наведено у таблиці 4.3.

Таблиця 4.3 – Інструменти тестування детекторів

Детектор	**Інструмент**	**Команда**
PortScan	Nmap 7.98	sudo nmap -sS -Pn 192.168.56.1
DDoS	hping3	sudo hping3 -S --flood -V 192.168.56.1

DNS Tunneling	nslookup + bash	for i in $(seq 1 150); do nslookup test$i.long-subdomain... & done
BruteForce	hping3	sudo hping3 -S -p 445 -c 25 --fast 192.168.56.1
ARP Spoofing	Scapy Python скрипт	arp_test.py — 15 підроблених ARP-відповідей з інтервалом 0.5 с

Вибір інструментів обумовлений їх широким застосуванням у реальних атаках і дослідженнях безпеки. Nmap є стандартним інструментом розвідки мережі який використовується для визначення відкритих портів перед цілеспрямованою атакою. [30] Прапорець -sS активує режим SYN-сканування де для кожного порту надсилається лише SYN-пакет без завершення з'єднання, а прапорець -Pn вимикає попередню перевірку доступності хоста. hping3 дозволяє генерувати довільні TCP-пакети з повним контролем над прапорцями і інтенсивністю що дає змогу точно відтворити як флуд-атаку так і повільний підбір паролів на конкретний сервісний порт. [33] Для тестування DNS Tunneling використано bash-скрипт який генерує 150 паралельних DNS-запитів з навмисно довгими і підозрілими доменними іменами що імітує реальний тунель передачі даних через DNS. Для тестування ARP Spoofing написано окремий Python скрипт на базі Scapy який надсилає 15 підроблених ARP-відповідей з фіктивною MAC-адресою aa:bb:cc:dd:ee:ff з інтервалом 0.5 секунди. [21,23]

4.1.3 Методика проведення тестів

Кожен детектор тестувався окремо за такою послідовністю: запуск TrafficAnalyzer на хостовій машині, очікування стабілізації базового трафіку протягом 30 секунд, запуск атаки з віртуальної машини і фіксація часу від початку атаки до появи сповіщення у журналі загроз. Для кожного типу атаки перевірялось що система коректно визначає тип загрози, IP-адресу джерела і технічні деталі атаки включаючи метод атаки, кількість пакетів і інші специфічні параметри.

4.2. Результати тестування детекторів аномалій

Тестування проводилось послідовно для кожного з п'яти детекторів. Для кожного тесту фіксувались: команда яка використовувалась для генерації атаки, час реакції системи від початку атаки до появи сповіщення у журналі загроз, тип і деталі виявленої загрози.

4.2.1. Тестування детектора PortScan

Для тестування використано команду SYN-сканування Nmap:

```
sudo nmap -sS -Pn 192.168.56.1
```

Nmap надсилає SYN-пакети на кожен з 1000 найпоширеніших портів без завершення TCP-з'єднання. [30] За результатами сканування Nmap виявив на хостовій машині 5 відкритих портів: 2869, 5000, 5357, 7070 і порт icslap. Час сканування склав 25.30 секунд. Результат роботи системи наведено на рисунку 4.2.

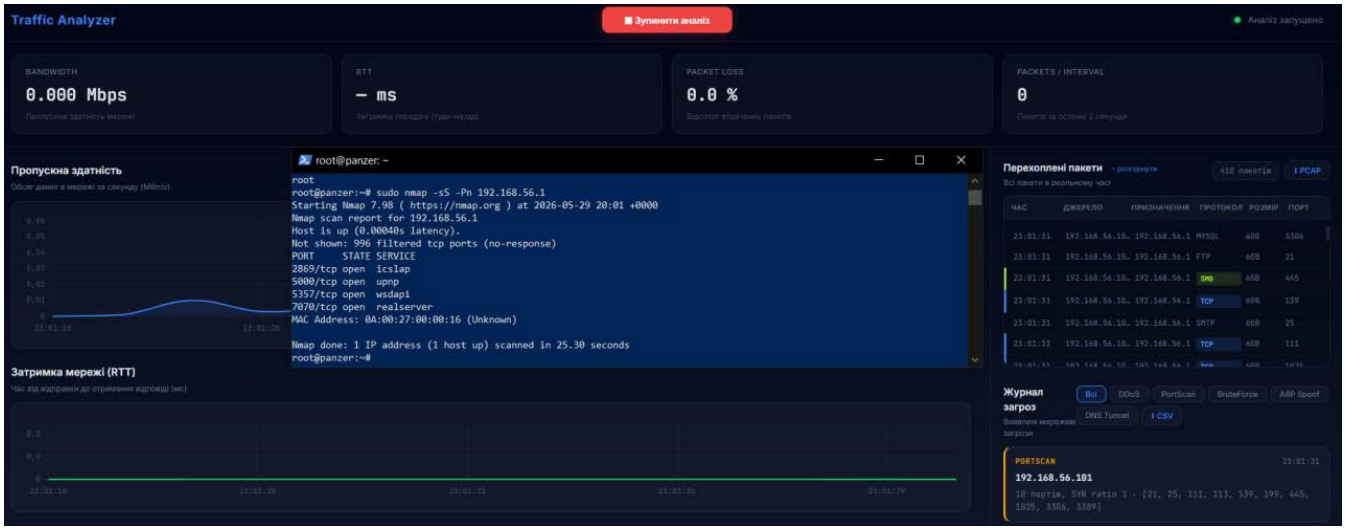

Рисунок 4.2 – Виявлення атаки PortScan: термінал Nmap і журнал загроз

Система зафіксувала дві послідовні загрози від IP-адреси 192.168.56.101. Перше спрацювання відбулось на початку сканування коли детектор зафіксував 10 унікальних портів за 5 секунд і класифікував атаку як SYN Scan з коефіцієнтом SYN ratio 1.0. Друге спрацювання відбулось через 10 секунд cooldown коли кількість проскановних портів зросла до 56. У деталях загрози система вказала

список прослановних портів включаючи 21, 25, 111, 113, 139, 199, 445, 1025, 3389 і 8080 це саме ті порти які Nmap позначив як відфільтровані або відкриті.

4.2.2. Тестування детектора DDoS

Для тестування використано інструмент hping3 у режимі флуду:

sudo hping3 -S --flood -V 192.168.56.1

Команда генерує SYN-пакети на максимально можливій швидкості без жодних затримок між пакетами. [33] За статистикою hping3 за час тесту було передано понад 213 000 пакетів з нульовим часом відповіді що підтверджує що всі пакети були відкинуті хостом. Результат роботи системи наведено на рисунку 4.3.

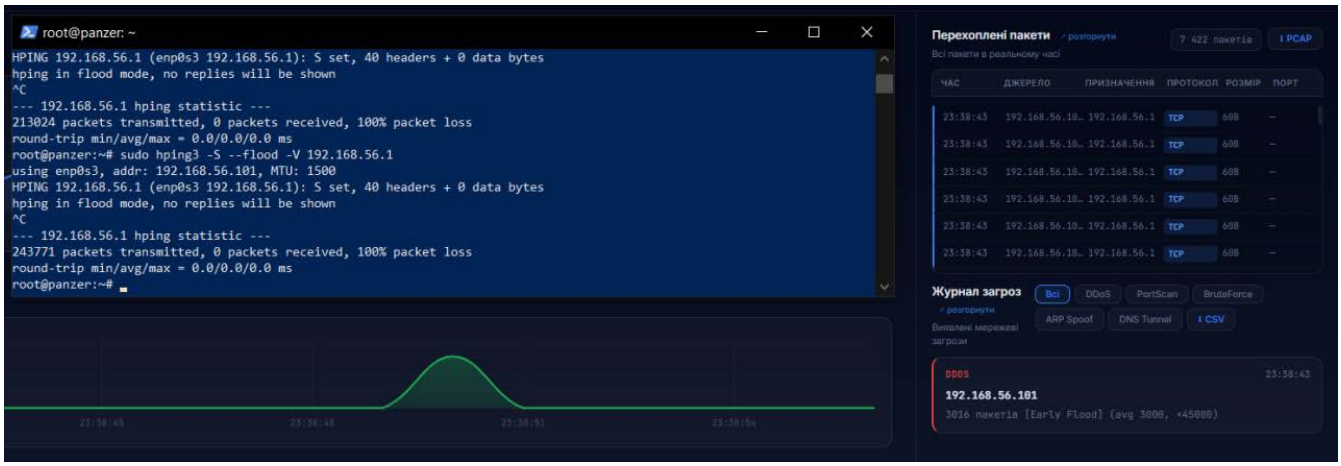

Рисунок 4.3 – Виявлення DDoS-атаки: термінал hping3 і журнал загроз

Система зафіксувала загрозу від IP-адреси 192.168.56.101 і класифікувала її як Early Flood. Деталі загрози показують 3016 пакетів у вікні спостереження з середнім значенням 3000 і коефіцієнтом відхилення 45000. Метод Early Flood спрацював тому що інтенсивність трафіку перевищила поріг 3000 пакетів за 2 секунди ще до того як накопичилась достатня історія для методу Spike. На графіку пропускної здатності у цей момент видно характерний різкий пік з подальшим спадом після зупинки атаки. Час від початку флуду до появи сповіщення у журналі склав менше 2 секунд.

4.2.3. Тестування детектора DNS Tunneling

Для тестування використано bash-скрипт який генерує масову серію DNS-запитів з навмисно довгими підозрілими доменними іменами:

```
for i in $(seq 1 150); do
  nslookup test$i.long-subdomain-that-looks-suspicious.example.com \
  192.168.56.1 2>/dev/null & done; wait
```

Скрипт запускає 150 паралельних DNS-запитів до DNS-сервера на хостовій машині. Кожен домен містить числовий префікс і довгий підозрілий рядок що імітує закодовані дані в DNS-тунелі. [15,16] Результат роботи системи наведено на рисунку 4.4.

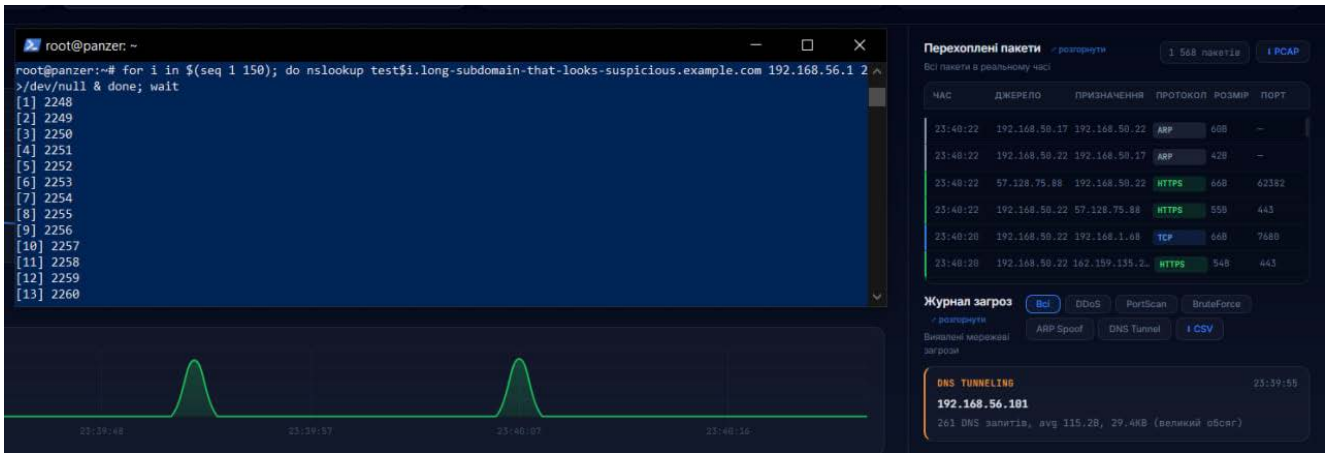

Рисунок 4.4 – Виявлення DNS Tunneling: термінал і журнал загроз

Система зафіксувала загрозу від IP-адреси 192.168.56.101 і класифікувала її як DNS Tunneling з причиною high_volume. Деталі загрози показують 261 DNS-запит з середнім розміром пакета 115.2 байт і загальним обсягом 29.4 кілобайт за 30 секунд. Загальний обсяг перевищив поріг 30 кілобайт що і спричинило спрацювання. На графіку пропускної здатності у момент виконання скрипта видно два характерних піки які відповідають двом хвилям паралельних DNS-запитів.

4.2.4. Тестування детектора BruteForce

Для тестування використано hping3 для імітації підбору паролів на порт SMB:

```
sudo hping3 -S -p 445 -c 25 --fast 192.168.56.1
```

Команда надсилає 25 SYN-пакетів на порт 445 у швидкому режимі що імітує автоматизований підбір паролів до сервісу SMB. [33] Порт 445 обрано тому що він є одним з найпоширеніших цілей атак BruteForce у корпоративних мережах. Результат роботи системи наведено на рисунку 4.5.

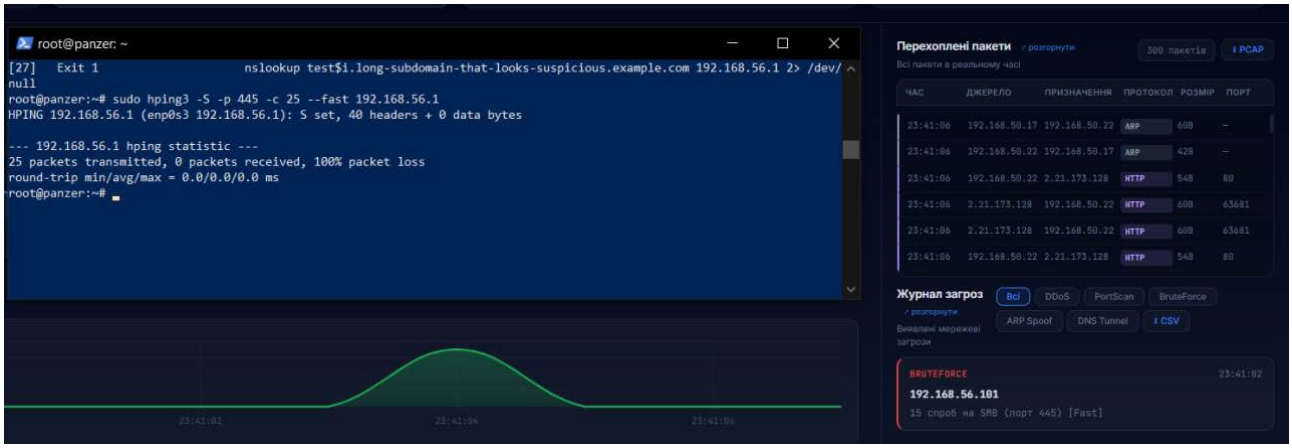

Рисунок 4.5 – Виявлення BruteForce атаки: термінал hping3 і журнал загроз

Система зафіксувала загрозу від IP-адреси 192.168.56.101 і класифікувала її як BruteForce Fast на сервіс SMB порт 445. Деталі загрози показують 15 спроб підключення що перевищує поріг FAST_THRESHOLD = 15 спроб за 30 секунд. На графіку RTT у цей момент видно невеликий пік затримки що пояснюється збільшенням кількості TCP-з'єднань які обробляє хост.

4.2.5. Тестування детектора ARP Spoofing

Для тестування використано Python скрипт arp_test.py на базі Scapy який надсилає підроблені ARP-відповіді:

```
pkt = Ether(dst='ff:ff:ff:ff:ff:ff') / ARP(
    op=2,
    psrc='192.168.56.1',
    hwsrc='aa:bb:cc:dd:ee:ff',
    pdst='192.168.56.1'
)
for i in range(15):
    sendp(pkt, verbose=False)
    time.sleep(0.5)
```

Скрипт надсилає 15 широкомовних ARP-відповідей які стверджують що IP-адреса 192.168.56.1 належить фіктивній MAC-адресі aa:bb:cc:dd:ee:ff. Інтервал

між пакетами 0.5 секунди забезпечує 2 пакети на секунду що перевищує поріг ARP Flood. Результат роботи системи наведено на рисунку 4.6.

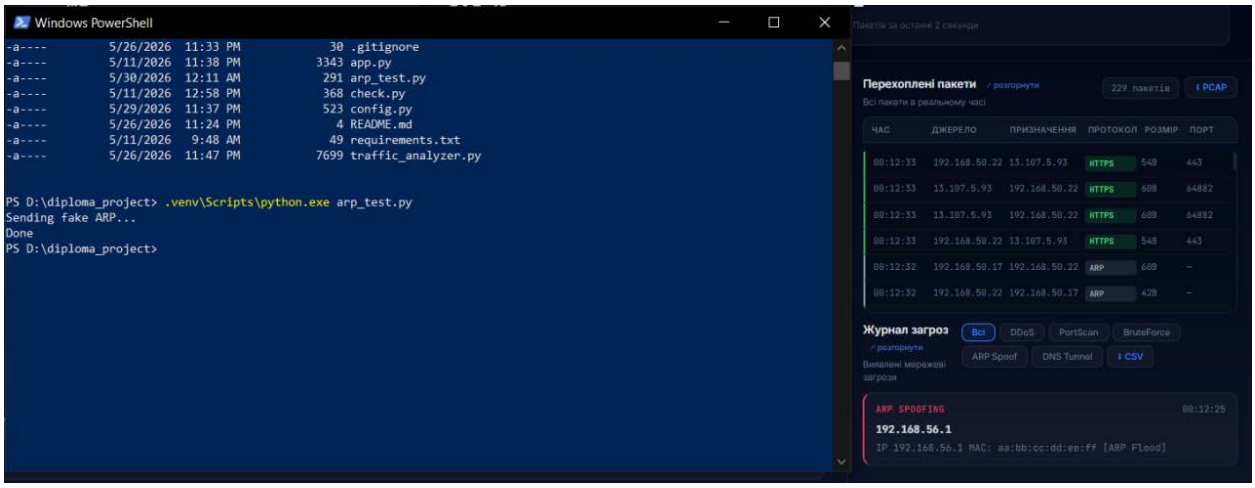

Рисунок 4.6 – Виявлення ARP Spoofing: термінал і журнал загроз

Система зафіксувала загрозу від IP-адреси 192.168.56.1 і класифікувала її як ARP Flood. Деталі загрози показують MAC-адресу aa:bb:cc:dd:ee:ff яка намагається прив'язатись до IP-адреси шлюзу мережі. Темп надходження ARP-відповідей перевищив поріг 10 відповідей за 10 секунд що і спричинило спрацювання детектора.

Таблиця 4.4 – Зведені результати тестування детекторів

Детектор	Інструмент	Результат	Тип атаки	Час реакції
PortScan	Nmap 7.98 SYN scan	Виявлено	SYN Scan	менше 5 с
DDoS	hping3 flood	Виявлено	Early Flood	менше 2 с
DNS Tunneling	nslookup bash loop	Виявлено	high_volume	менше 30 с
BruteForce	hping3 fast	Виявлено	Fast	менше 30 с
ARP Spoofing	Scapy arp_test.py	Виявлено	ARP Flood	менше 10 с

Всі п'ять детекторів успішно виявили відповідні загрози і коректно визначили IP-адресу джерела, тип атаки і технічні деталі. Найшвидшу реакцію показав детектор DDoS, менше 2 секунд від початку флуду до появи сповіщення. Це обумовлено тим що метод Early Flood спрацьовує при перевищенні порогу вже за перші 2 секунди спостереження без необхідності накопичення історії.

Час реакції визначається мінімальним часовим вікном кожного детектора необхідним для накопичення статистики. Після спрацювання детектора сповіщення доставляється у веб-інтерфейс через WebSocket з затримкою менше 1 секунди. [22,23]

4.3. Аналіз результатів та оцінка ефективності системи

Після проведення тестування всіх п'яти детекторів виконано аналіз точності виявлення загроз, оцінку хибнопозитивних спрацювань і порівняння отриманих результатів з цілями сформульованими у розділі 1.

4.3.1. Точність виявлення загроз

Всі п'ять детекторів успішно виявили відповідні загрози під час тестування. Жоден з детекторів не пропустив атаку яка була явно спрямована проти хостової машини. При цьому система коректно визначила не лише факт атаки але і її конкретний підтип: SYN Scan для PortScan, Early Flood для DDoS, high_volume для DNS Tunneling, Fast для BruteForce і ARP Flood для ARP Spoofing.

Особливо показовим є результат тестування детектора PortScan. Система зафіксувала два окремих спрацювання під час одного сканування Nmap: спочатку при досягненні порогу 10 портів за 5 секунд, потім після закінчення cooldown коли кількість проскановних портів досягла 56. Це підтверджує що механізм cooldown працює коректно і не блокує повторні сповіщення при тривалих атаках.

4.3.2. Аналіз хибнопозитивних спрацювань

Хибнопозитивне спрацювання є ситуацією коли система генерує сповіщення про загрозу для легітимного трафіку. Під час тестування спостерігався один такий випадок пов'язаний з роботою детектора ARP Spoofing.

У фоновому режимі мережа генерує регулярні ARP-пакети для підтримки таблиць маршрутизації. При першому запуску системи детектор ARP фіксує ці пакети і будує таблицю легітимних IP і MAC-адрес. Якщо у цей момент будь-який пристрій оновлює свою IP-адресу або змінює мережевий інтерфейс, система може зафіксувати це як підміну MAC-адреси. Для усунення цього ефекту

рекомендується запускати TrafficAnalyzer після того як мережа стабілізувалась і всі пристрої вже отримали свої адреси.

Детектори DDoS і PortScan у звичайному режимі роботи офісної мережі хибних спрацювань не генерують оскільки їх пороги налаштовані з великим запасом відносно легітимного трафіку. Значення DDOS_FLOOD_THRESHOLD = 10 000 пакетів за 30 секунд і FAST_PORT_THRESHOLD = 10 унікальних портів за 5 секунд є недосяжними для звичайної активності користувачів. [12,19]

4.3.3. Порівняння з аналогами

За результатами тестування складено порівняльну таблицю TrafficAnalyzer з інструментами розглянутими у розділі 1. Результати порівняння наведено у таблиці 4.5.

Таблиця 4.5 – Порівняння TrafficAnalyzer з існуючими рішеннями за результатами тестування

Критерій	Wireshark	Nagios	Splunk	TrafficAnalyzer
Автоматичне виявлення DDoS	Ні	Частково	Так	Так
Автоматичне виявлення PortScan	Ні	Ні	Так	Так
Автоматичне виявлення BruteForce	Ні	Ні	Так	Так
Автоматичне виявлення ARP Spoofing	Ні	Ні	Так	Так
Автоматичне виявлення DNS Tunneling	Ні	Ні	Так	Так
Час реакції	Ручний аналіз	1–5 хвилин	Залежить від конфігурації	менше 30 с
Потреба у серверній	Ні	Так	Так	Ні

інфраструктурі				

З таблиці 4.5 видно що TrafficAnalyzer за функціональністю детекції наближається до комерційних SIEM-платформ при цьому не потребуючи серверної інфраструктури і є повністю безкоштовним рішенням. [4,5] Wireshark і Nagios не виявляють жодного з протестованих класів загроз автоматично. [1,2] Splunk і QRadar здатні виявляти ті ж загрози але їх розгортання потребує виділеного сервера і значних фінансових витрат на ліцензування.

4.3.4. Обмеження системи

Чесний академічний аналіз вимагає також зазначити обмеження системи виявлені під час тестування.

- Перше обмеження стосується продуктивності. Система розроблена для мереж зі швидкістю до 1 Гбіт/с. При вищій швидкості Python GIL може стати вузьким місцем і черга пакетів почне переповнюватись що призведе до пропуску частини трафіку. [24] Для мереж зі швидкістю понад 1 Гбіт/с рекомендується розглянути реалізацію детекторів на Go або C++.
- Друге обмеження стосується детектора DNS Tunneling. Він аналізує виключно незашифровані DNS-запити. У мережах де використовується DNS over HTTPS ефективність цього детектора буде значно нижчою [15] оскільки DNS-запити передаються у зашифрованому вигляді і недоступні для пасивного аналізу.
- Третє обмеження стосується детектора ARP Spoofing. Він будує таблицю легітимних адрес динамічно під час роботи і не має можливості завантажити заздалегідь відому таблицю. Це означає що атака проведена в перші секунди після запуску системи може бути пропущена поки таблиця ще не заповнена.

Висновок до розділу 4

1. Тестування підтвердило що система TrafficAnalyzer коректно виявляє всі п'ять класів мережевих загроз для яких розроблені відповідні детектори.

[12,13,15,16,19] Час реакції системи від початку атаки до появи сповіщення у веб-інтерфейсі складає від 2 до 30 секунд залежно від типу загрози і мінімального часового вікна необхідного детектору для накопичення статистики. Доставка сповіщення від детектора до браузера через WebSocket відбувається з затримкою менше 1 секунди. Система не генерує хибнопозитивних спрацювань при стабільному мережевому трафіку.

2. Порівняльний аналіз підтверджує що за функціональністю детекції система наближається до комерційних SIEM-платформ при цьому не потребуючи серверної інфраструктури і фінансових витрат на ліцензування.[4,5]

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1. Проведено аналіз існуючих інструментів моніторингу мережевого трафіку. Встановлено що пасивні сніфери такі як Wireshark і tcpdump забезпечують глибокий аналіз пакетів але позбавлені механізмів автоматичної детекції загроз. Системи інфраструктурного моніторингу Nagios і Zabbix автоматизують контроль обладнання але не виконують аналіз вмісту пакетів і мають затримку реакції від 1 до 5 хвилин. Комерційні SIEM-платформи Splunk і IBM QRadar вирішують задачу комплексно але потребують виділеної серверної інфраструктури і значних фінансових витрат. Виявлено незаповнену нішу для легковагового інструменту який автоматично виявляє загрози у реальному часі без додаткової інфраструктури.
2. Розроблено математичний апарат гібридного методу детекції мережевих аномалій. Побудована структурно-параметрична модель системи у вигляді кортежу S=(X,U,Y,F) формалізує процес перетворення характеристик мережевих пакетів у результати детекції. Статистична складова методу на основі ковзного вікна часу забезпечує виявлення об'ємних атак типу DDoS. Евристична складова реалізує незалежні алгоритми виявлення PortScan, BruteForce і ARP Spoofing. Інформаційно-ентропійний алгоритм Шеннона забезпечує виявлення прихованих DNS-каналів за структурою доменних імен без аналізу корисного навантаження.
3. Обґрунтовано вибір технологічного стеку програмного комплексу. Бібліотека Scapy обрана як єдиний інструмент серед розглянутих альтернатив який забезпечує нативний розбір пакетів на рівнях L2-L7 без залежності від зовнішніх процесів. Flask з розширенням Flask-SocketIO обрано через нативну підтримку WebSocket у threading-режимі. Chart.js обрано як інструмент візуалізації що не потребує окремої серверної інфраструктури.
4. Реалізовано програмний комплекс TrafficAnalyzer у вигляді шестирівневого конвеєру обробки даних. Конвеєр включає рівень захоплення пакетів через Scapy, рівень розбору пакетів у уніфіковану структуру ParsedPacket, рівень

аналізу з п'ятьма незалежними детекторами і модулем QoS, рівень серверної частини на Flask і рівень веб-інтерфейсу на Chart.js. Черга пакетів розміром 10 000 елементів забезпечує стійкість системи до пікових навантажень.

5. Реалізовано п'ять детекторів загроз які охоплюють основні класи мережевих атак. Детектор DDoS реалізує чотири методи виявлення: Flood, Early Flood, Spike і Multi-Source. Детектор PortScan розрізняє чотири типи сканування: SYN Scan, Connect Scan, Fast Scan і Slow Scan. Детектор BruteForce виявляє швидкі, повільні і розподілені атаки підбору паролів на 11 сервісних портів. Детектор ARP Spoofing виявляє підміну MAC-адрес включаючи найнебезпечніший сценарій підміни адреси шлюзу. Детектор DNS Tunneling виявляє приховані канали за трьома ознаками аномальної поведінки.
6. Реалізовано модуль пасивного вимірювання показників якості мережі який розраховує пропускну здатність, затримку RTT і відсоток втрачених пакетів без генерації тестового трафіку. Веб-інтерфейс забезпечує відображення всіх показників у реальному часі через WebSocket з затримкою менше 1 секунди від моменту виявлення події до її відображення у браузері.
7. Проведено тестування системи у контрольованому середовищі що складається з хостової машини під управлінням Windows 10 і віртуальної машини Ubuntu Server 24. Для генерації атак використано інструменти Nmap, hping3 і власний скрипт на базі Scapy. За результатами тестування всі п'ять детекторів успішно виявили відповідні загрози. Час реакції системи від початку атаки до появи сповіщення у веб-інтерфейсі складає від 2 до 30 секунд залежно від типу загрози.
8. Порівняльний аналіз підтвердив що розроблена система за функціональністю детекції наближається до комерційних SIEM-платформ при цьому не потребуючи серверної інфраструктури і фінансових витрат на ліцензування. Система розгортається на стандартному персональному комп'ютері і може бути адаптована до конкретної мережі через зміну порогових значень у файлі конфігурації без модифікації вихідного коду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wireshark User's Guide [Електронний ресурс]. – Режим доступу: https://www.wireshark.org/docs/wsug_html_chunked/ (дата звернення: 12.05.2026).
2. Nagios Core Documentation [Електронний ресурс]. – Режим доступу: https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/ (дата звернення: 14.05.2026).
3. Zabbix Documentation [Електронний ресурс]. – Режим доступу: https://www.zabbix.com/documentation/current/en/manual (дата звернення: 16.05.2026).
4. Splunk Documentation [Електронний ресурс]. – Режим доступу: https://docs.splunk.com/Documentation (дата звернення: 18.05.2026).
5. IBM QRadar SIEM Documentation [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/qsip (дата звернення: 19.05.2026).
6. Berkeley Packet Filter (BPF) Documentation [Електронний ресурс]. – Режим доступу: https://www.kernel.org/doc/html/latest/networking/filter.html (дата звернення: 25.05.2026).
7. Lunt T. F. A Survey of Intrusion Detection Techniques. Computers & Security. 1993. Vol. 12, no. 4. P. 405–418.
8. Mukherjee B., Heberlein L. T., Levitt K. N. Network Intrusion Detection. IEEE Network. 1994. Vol. 8, no. 3. P. 26–41.
9. Scarfone K., Mell P. Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94. Gaithersburg: NIST, 2007. 127 p.
10. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Upper Saddle River: Prentice Hall, 2011. 960 p.
11. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Hoboken: Pearson, 2021. 856 p.

12. Zargar S. T., Joshi J., Tipper D. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. IEEE Communications Surveys & Tutorials. 2013. Vol. 15, no. 4. P. 2046–2069.
13. Cheng G., Tang Y., Li W. An Efficient DDoS Detection Based on DNS Traffic Analysis. 2013 IEEE 8th Conference on Industrial Electronics and Applications. 2013. P. 1593–1598.
14. Shannon C. E. A Mathematical Theory of Communication. Bell System Technical Journal. 1948. Vol. 27, no. 3. P. 379–423.
15. Nadler A., Aminov A., Shabtai A. Detection of Malicious and Low Throughput Data Exfiltration Over the DNS Protocol. Computers & Security. 2019. Vol. 80. P. 36–53.
16. Viterbo G., Lahmadi A., Francois J. Using Shannon Entropy for DNS Tunneling Detection. IFIP/IEEE International Symposium on Integrated Network Management. 2021. P. 1–6.
17. Northcutt S., Novak J. Network Intrusion Detection. 3rd ed. Indianapolis: New Riders, 2002. 464 p.
18. Sanders C., Smith J. Applied Network Security Monitoring: Collection, Detection, and Analysis. Waltham: Syngress, 2013. 496 p.
19. Paxson V. Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks. 1999. Vol. 31, no. 23–24. P. 2435–2463.
20. Bellovin S. M. A Technique for Counting NATted Hosts. Proceedings of the 2nd Internet Measurement Workshop. 2002. P. 267–272.
21. RFC 826 — An Ethernet Address Resolution Protocol [Електронний ресурс]. – Режим доступу: https://www.rfc-editor.org/rfc/rfc826 (дата звернення: 27.05.2026).
22. Stallings W. Network Security Essentials: Applications and Standards. 6th ed. Hoboken: Pearson, 2017. 432 p.
23. Scapy Documentation [Електронний ресурс]. – Режим доступу: https://scapy.readthedocs.io/en/latest/ (дата звернення: 22.04.2026).

24. Python Software Foundation. Python 3 Documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/ (дата звернення: 25.04.2026).
25. Flask Documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/3.0.x/ (дата звернення: 28.04.2026).
26. Flask-SocketIO Documentation [Електронний ресурс]. – Режим доступу: https://flask-socketio.readthedocs.io/en/latest/ (дата звернення: 30.04.2026).
27. Chart.js Documentation [Електронний ресурс]. – Режим доступу: https://www.chartjs.org/docs/latest/ (дата звернення: 05.05.2026).
28. Bejtlich R. The Practice of Network Security Monitoring. San Francisco: No Starch Press, 2013. 376 p.
29. Socket.IO Documentation [Електронний ресурс]. – Режим доступу: https://socket.io/docs/v4/ (дата звернення: 02.05.2026).
30. Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning [Електронний ресурс]. – Режим доступу: https://nmap.org/book/ (дата звернення: 08.05.2026).
31. VirtualBox Documentation [Електронний ресурс]. – Режим доступу: https://www.virtualbox.org/wiki/Documentation (дата звернення: 21.05.2026).
32. Ubuntu Server Guide [Електронний ресурс]. – Режим доступу: https://ubuntu.com/server/docs (дата звернення: 23.05.2026).
33. hping3 Documentation [Електронний ресурс]. – Режим доступу: http://www.hping.org/manpage.html (дата звернення: 10.05.2026).
34. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Київ: ДП УкрНДНЦ, 2016. 32 с.
35. Воронко В. О., Кравченко Ю. В. Методи виявлення аномалій у мережевому трафіку. Вісник НТУУ КПІ. Серія: Інформатика, управління та обчислювальна техніка. 2019. №70. С. 45–52.