ЖИТОМИРСЬКИЙ ВІЙСЬКОВИЙ ІНСТИТУТУ ІМЕНІ С.П.КОРОЛЬОВА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

здобувача вищої освіти ступеня «бакалавр» заочної форми навчання
за спеціальністю «Кібербезпека»

Тема: «ПРОГРАМНИЙ ДОДАТОК КОНТРОЛЮ СТАНУ МЕРЕЖІ ТА ВИЯВЛЕННЯ НЕСАНКЦІОНОВАНОГО ДОСТУПУ ДО НЕЇ»

Виконавець: Олег БІЛОБОЦЬКИЙ

м. Житомир

2026

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох основних розділів, загальних висновків, переліку джерел посилання та додатків, містить 57 сторінок основного тексту, 27 рисунків, 6 таблиць, використано 45 джерел. Загальний обсяг роботи складає 76 сторінок.

Об’єктом дослідження є процес контролю стану локального мережевого середовища та виявлення потенційно несанкціонованого доступу.

Предметом дослідження є методи та засоби моніторингу локальних мереж і виявлення підозрілих мережевих пристроїв.

Метою кваліфікаційної роботи є розроблення програмного додатку контролю стану локальної мережі та виявлення потенційно несанкціонованого доступу.

У роботі проведено аналіз сучасних мережевих кіберзагроз і існуючих засобів мережевого моніторингу. На основі проведеного аналізу розроблено функціональну структуру програмного додатку, алгоритми збору мережевих даних, аналізу пристроїв, виявлення аномалій, журналювання подій і формування статистичних показників.

Реалізовано програмний додаток, що забезпечує виявлення активних мережевих пристроїв, аналіз змін *IP*- та *MAC*-ідентифікаторів, виявлення потенційно підозрілих вузлів і ведення журналу подій.

Працездатність розробленого програмного додатку підтверджено методом натурного моделювання у реальному локальному мережевому середовищі.

Розроблений програмний додаток може використовуватися у навчальних, тестових і локальних корпоративних мережах.

Ключові слова: ЛОКАЛЬНА МЕРЕЖА, МЕРЕЖЕВИЙ МОНІТОРИНГ, НЕСАНКЦІОНОВАНИЙ ДОСТУП, КІБЕРБЕЗПЕКА, *IP*-АДРЕСА, *MAC*-АДРЕСА, ВИЯВЛЕННЯ АНОМАЛІЙ, КОНТРОЛЬ СТАНУ МЕРЕЖІ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІКС	–	Інформаційно комунікаційна система
ІС	–	Інформаційна система
ШПЗ	–	Шкідливе програмне забезпечення
MITM	–	(*Man-in-the-Middle*) - атака типу «людина посередині»
ARP-spoofing		Мережева атака в локальних мережах
ISO/IEC		Стандарти інформаційної безпеки
NIST	–	*National Institute of Standards and Technology* — Національний інститут стандартів і технологій США.
НД ТЗІ		Нормативні документи системи технічного захисту інформації
ARP		*Address Resolution Protocol* — протокол визначення відповідності між *IP*-адресою та *MAC*-адресою в локальній мережі.
CIDR	–	*Classless Inter-Domain Routing* — метод адресації та опису діапазонів *IP*-адрес.
DHCP		*Dynamic Host Configuration Protocol* — протокол автоматичного призначення мережевих параметрів пристроям
IP		*Internet Protocol* — мережевий протокол адресації вузлів
MAC		*Media Access Control* — унікальний фізичний ідентифікатор мережевого адаптера
SQLite		Вбудована файлова система керування базами даних
NPCAP		*Nmap Packet Capture* — драйвер доступу до пакетного рівня мережевого інтерфейсу.

ВСТУП

У сучасних умовах активного розвитку інформаційно-комунікаційних систем (ІКС) локальні комп’ютерні мережі є невід’ємною складовою функціонування навчальних закладів, підприємств, державних установ та інших організацій. Водночас зростання кількості підключених пристроїв, використання бездротових технологій та підвищення складності мережевої інфраструктури створюють додаткові ризики для інформаційної безпеки.

Однією з актуальних проблем кібербезпеки є несанкціоноване підключення пристроїв до локального мережевого середовища, що може призводити до витоку інформації, порушення цілісності мережевої інфраструктури, створення умов для реалізації мережевих атак або дестабілізації роботи ІКС. Особливу небезпеку становлять випадки, коли сторонні або підозрілі пристрої маскуються під легітимні мережеві вузли шляхом зміни мережевих ідентифікаторів або використання типових параметрів локального сегмента.

Існуючі програмні засоби мережевого моніторингу часто орієнтовані на корпоративне застосування, мають надлишкову функціональність, складність налаштування або значні вимоги до ресурсів. У навчальних, тестових або малих локальних мережах доцільним є використання легких автономних програмних засобів, адаптованих до завдань контролю стану мережевого середовища та виявлення потенційно несанкціонованого доступу.

У зв’язку з цим тема розроблення програмного додатку контролю стану мережі та виявлення несанкціонованого доступу до неї є актуальною.

РОЗДІЛ 1. ОБГРУНТУВАННЯ НЕОБХІДНОСТІ РОЗРОБЛЕННЯ ДОДАТКУ КОНТРОЛЮ СТАНУ МЕРЕЖІ

1.1 Аналіз сучасних мережевих кіберзагроз та проблеми несанкціонованого доступу

Комп’ютерна мережа організації є не лише засобом передавання даних між пристроями, але і середовищем функціонування більшості інформаційних сервісів. Через це вона фактично виступає центральним елементом інформаційної інфраструктури, порушення роботи якого може вплинути на всі бізнес-процеси або діяльність установи.

На відміну від попередніх років, сучасні мережеві атаки рідко обмежуються одноразовим втручанням. У більшості випадків вони мають багатоетапний характер: від збору інформації про мережу до закріплення у внутрішньому середовищі та подальшого прихованого використання отриманого доступу. Саме тому контроль стану мережі є одним із ключових інструментів раннього виявлення загроз.

Серед найбільш характерних типів атак, що зустрічаються в локальних та корпоративних мережах, можна виділити такі, що представлені на рис. 1.1.

Першою поширеною категорією є сканування портів. Такий тип активності використовується для визначення відкритих мережевих служб, активних вузлів та потенційних точок доступу до мережі [22]. Сканування портів часто є підготовчим етапом атаки, оскільки дозволяє зловмиснику отримати первинну інформацію про структуру мережі та доступні точки входу.

Наступним поширеним типом є підбір облікових даних, або *brute force*-атаки. Їх сутність полягає у багаторазових спробах автентифікації з використанням різних комбінацій логінів і паролів [30]. У локальних та корпоративних мережах такі атаки можуть бути спрямовані на служби

віддаленого доступу, веб-інтерфейси адміністрування, поштові сервіси або внутрішні інформаційні ресурси.

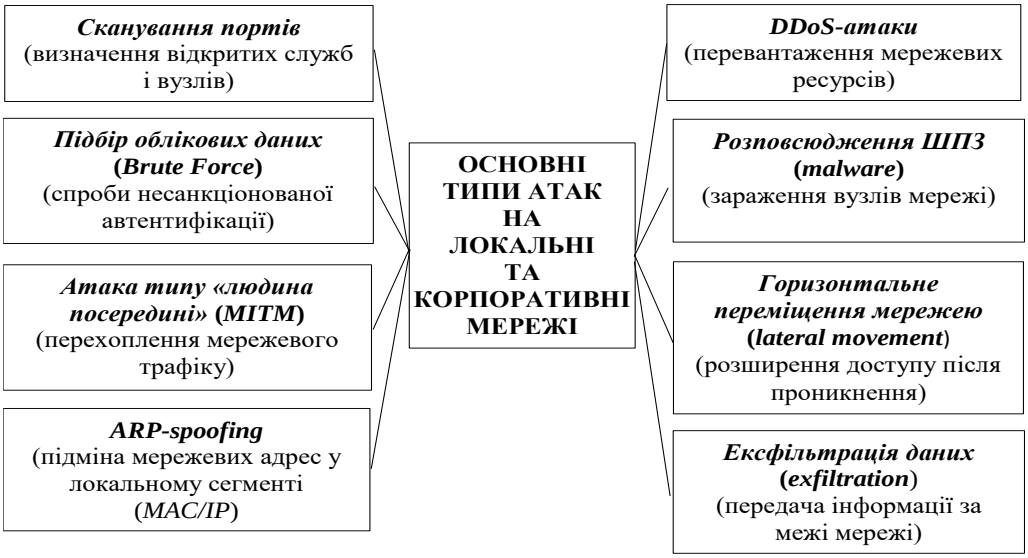

Рис. 1.1 Основні типи атак на локальні та корпоративні мережі

Окрему небезпеку становлять атаки типу «людина посередині» (*MITM*). Під час такої атаки зловмисник перехоплює мережевий трафік між двома вузлами мережі, що може призвести до витоку облікових даних, службової інформації або підміни переданих даних. *MITM*-атаки особливо небезпечні у незахищених або недостатньо сегментованих локальних мережах [31].

Близьким за механізмом є *ARP-spoofing*, який базується на підміні відповідностей між *IP*-адресами та *MAC*-адресами мережевих пристроїв у локальному сегменті мережі. Унаслідок цього трафік користувача може бути перенаправлений через пристрій зловмисника. Такий тип атаки часто використовується для перехоплення трафіку або підготовки до атаки типу *MITM*. [17]

До атак на доступність належать *DDoS*-атаки, метою яких є перевантаження мережевих ресурсів, серверів або каналів зв’язку великою кількістю запитів. Наслідком таких атак є порушення нормального функціонування сервісів, зниження продуктивності мережі або повна недоступність окремих інформаційних ресурсів [32].

Ще однією поширеною загрозою є розповсюдження шкідливого програмного забезпечення (ШПЗ) (*malware*) в межах мережі. Після потрапляння на один із вузлів ШПЗ може поширюватися на інші пристрої, використовувати мережеві ресурси для зв’язку з командним сервером або виконувати дії, спрямовані на викрадення чи пошкодження інформації [33].

Після отримання початкового доступу зловмисник може здійснювати горизонтальне переміщення мережею (*lateral movement*). Це передбачає поступове розширення контролю над іншими вузлами, використання службових облікових записів, мережевих протоколів і внутрішніх сервісів. Такий етап є особливо небезпечним, оскільки дії зловмисника можуть маскуватися під звичайну активність користувачів або адміністраторів [34].

Завершальним або одним із ключових етапів багатьох атак є ексфільтрація даних (*exfiltration*), тобто несанкціоноване передавання інформації за межі мережі організації. Вона може здійснюватися через стандартні протоколи, хмарні сервіси, зашифровані канали або приховані мережеві з’єднання. За відсутності контролю мережевої активності така передача даних може залишатися непоміченою протягом тривалого часу [35].

Таким чином, наведені на рис. 1.1 типи атак охоплюють як підготовчі дії зловмисника, так і етапи безпосереднього проникнення, закріплення, поширення в мережі та виведення даних. Це підтверджує необхідність використання програмного додатку, здатного контролювати стан мережі, виявляти підозрілу активність і фіксувати ознаки несанкціонованого доступу.

Останні роки характеризуються суттєвими змінами у характері кіберзагроз. Однією з причин цього є активне впровадження віддалених форм роботи, використання хмарних сервісів та збільшення кількості підключених до мережі пристроїв.

Як показано на рис. 1.2, протягом 2020–2025 років спостерігається зростання кількості атак на облікові записи користувачів та периферійні мережеві сервіси, активне використання автоматизованих засобів проведення атак, поширення програм-вимагачів, а також збільшення масштабів *DDoS*-

атак. Крім того, характерною тенденцією є орієнтація кіберзагроз на об’єкти критичної інформаційної інфраструктури та зростання кількості кіберінцидентів у державних і корпоративних мережах. Це підтверджує необхідність впровадження засобів постійного контролю стану мережі.

Рис. 1.2 — Основні тенденції розвитку мережевих кіберзагроз у 2020–2025 роках

Однією з найбільш помітних тенденцій є зростання кількості атак на механізми автентифікації. Обліковий запис користувача фактично став новою точкою входу до інформаційної системи. Зловмисники активно використовують автоматизовані засоби перевірки паролів та облікових записів.

Іншою важливою тенденцією є збільшення кількості атак, спрямованих на периферійні компоненти мережевої інфраструктури. До таких компонентів належать шлюзи віддаленого доступу, сервери передачі файлів та веб-інтерфейси адміністрування.

Крім того, спостерігається постійне зростання фінансових втрат від кіберінцидентів. Це свідчить про підвищення рівня організованості атак та їх економічної мотивації.

Особливу увагу слід приділити ситуації в Україні, де кількість зафіксованих кіберінцидентів протягом останніх років демонструє стабільну тенденцію до збільшення. Це пов’язано як із загальносвітовими тенденціями цифровізації, так і з безпековою ситуацією держави.

Ще однією характерною ознакою сучасного періоду є широке застосування автоматизованих засобів проведення атак. Це значно скорочує час підготовки вторгнення та підвищує ефективність пошуку вразливих вузлів мережі.

Відсутність системного моніторингу стану мережі створює передумови для прихованого функціонування зловмисника в ІКС протягом тривалого часу. У більшості випадків атаки на комп’ютерні мережі мають багатоетапний характер, і саме на початкових етапах їх реалізації можливо своєчасно виявити ознаки несанкціонованого доступу за умови використання відповідних засобів контролю мережевої активності.

До основних наслідків відсутності мережевого моніторингу належать (рис. 1.3):

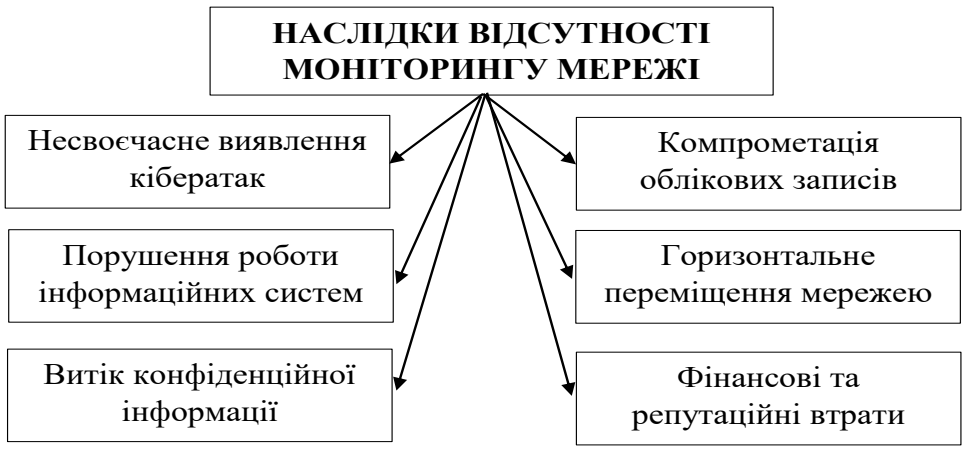

Рис. 1.3 — Основні наслідки відсутності моніторингу стану мережі

Практика кібербезпеки останніх років підтверджує, що значна частина масштабних інцидентів стала можливою саме через відсутність своєчасного контролю стану мережевої інфраструктури.

Показовим прикладом є кібератака на компанію *SolarWinds* у 2020 році. У результаті компрометації програмного забезпечення *Orion* зловмисники отримали доступ до внутрішніх мереж державних установ та великих корпорацій. Особливістю цього інциденту стало тривале приховане перебування зловмисників у мережевому середовищі без своєчасного виявлення ознак втручання.

Іншим прикладом є атака із використанням програми-вимагача *Colonial Pipeline* у 2021 році, яка призвела до тимчасового припинення роботи паливної інфраструктури США. Інцидент продемонстрував критичну залежність інфраструктурних об’єктів від стабільності функціонування інформаційних систем (ІС) та необхідність постійного контролю мережевої активності.

Особливо показовими є кіберінциденти, пов’язані з атакою на супутникову мережу *KA-SAT* компанії *Viasat* у 2022 році (рис. 1.4).

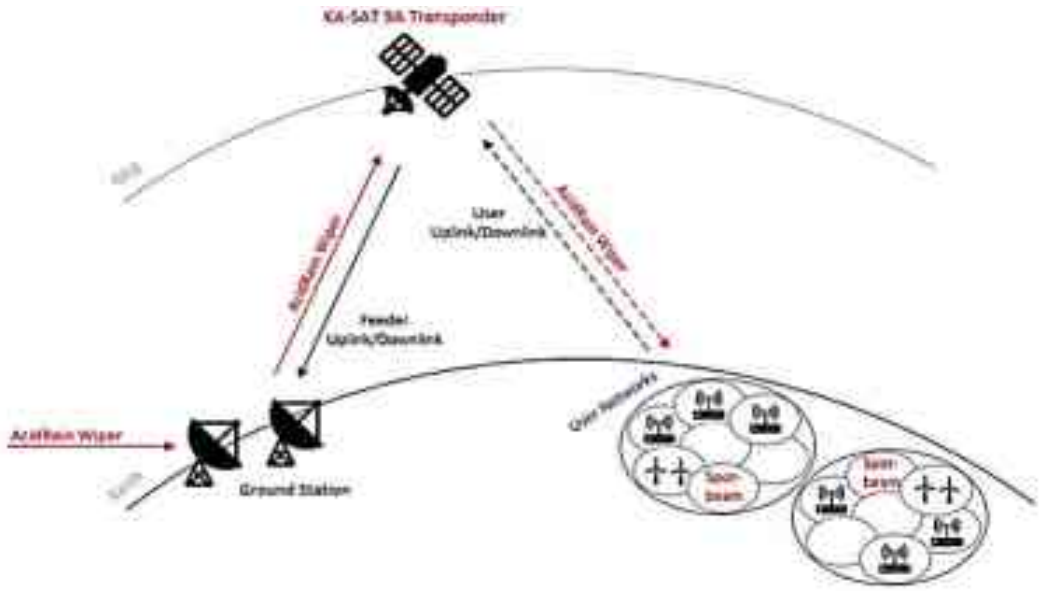

Рис. 1.4 *Wiper* (вірус-знищувач) «падає дощем» з супутника

Унаслідок втручання було порушено роботу супутникових модемів у різних регіонах Європи, зокрема в Україні. Цей інцидент підтвердив, що

мережеві атаки можуть впливати не лише на окремі організації, але й на функціонування критичної інфраструктури державного рівня [2, 3, 4].

Для України проблема забезпечення контролю стану мережевої інфраструктури є особливо актуальною. За даними урядової команди реагування на комп'ютерні надзвичайні події *CERT-UA*, кількість кіберінцидентів у державному секторі України демонструє стабільну тенденцію до зростання. Значна частина таких інцидентів пов'язана зі спробами несанкціонованого доступу до державних інформаційних ресурсів та об'єктів критичної інфраструктури.

Одним із найбільш показових прикладів кіберінцидентів, що підтверджують необхідність контролю стану мережевої інфраструктури, є атаки на енергосистему України у 2015 та 2016 роках. У грудні 2015 року внаслідок скоординованої кібератаки було порушено роботу електропостачання у декількох регіонах України, що призвело до відключення електроенергії приблизно для 230 тисяч споживачів. Атака здійснювалася шляхом попередньої компрометації корпоративних мереж енергетичних компаній із використанням шкідливого програмного забезпечення *BlackEnergy* та подальшого отримання доступу до систем диспетчерського керування електромережею.

Особливістю цього інциденту стало те, що зловмисники протягом тривалого часу залишалися непоміченими у внутрішніх мережах операторів енергетичної інфраструктури, здійснювали мережеву розвідку, збір облікових даних користувачів та поступово отримували контроль над елементами систем керування підстанціями. Це підтверджує критичну роль засобів моніторингу мережевої активності для своєчасного виявлення ознак вторгнення.

Наступного року було зафіксовано повторну атаку на енергетичну інфраструктуру України, яка призвела до тимчасового відключення електропостачання частини міста Києва. У цьому випадку використовувалося спеціалізоване шкідливе програмне забезпечення *Industroyer*, орієнтоване безпосередньо на системи автоматизованого керування енергетичними

підстанціями. Зловмисники отримали можливість дистанційно керувати комутаційним обладнанням та виводити з ладу елементи енергомережі.

Крім того, у 2022 році українським фахівцям із кібербезпеки вдалося запобігти ще одній спробі кібератаки на енергосистему держави із використанням модифікованого шкідливого програмного забезпечення типу *Industroyer*. Цей інцидент підтвердив, що об’єкти енергетичної інфраструктури залишаються пріоритетною ціллю сучасних кібероперацій.

На рис. 1.5 показана реалізація кібератаки на енергетичну інфраструктуру, яка здійснюється поетапно та передбачає початкову компрометацію корпоративної мережі організації, отримання доступу до облікових записів операторів і подальше втручання у роботу систем диспетчерського керування електромережею.

Рис. 1.5 — Приклад реалізації кібератаки на енергетичну інфраструктуру через мережеве проникнення

Ще одним прикладом масштабного кіберінциденту є атака на програмне забезпечення *MOVEit Transfer* у 2023 році, яка призвела до масштабного витоку конфіденційних даних організацій у різних країнах світу. Використання уразливості у сервісі передавання файлів дозволило зловмисникам отримати доступ до конфіденційної інформації без своєчасного виявлення ознак вторгнення.

Таким чином, аналіз реальних кіберінцидентів підтверджує, що відсутність системного контролю мережевої активності суттєво підвищує ризик компрометації інформаційних ресурсів та порушення функціонування ІКС [15, 16].

1.2 Аналіз існуючих рішень моніторингу мереж і їхніх обмежень

Сучасні підходи до забезпечення мережевої безпеки базуються на вимогах міжнародних та національних нормативних документів. Зокрема стандарт *ISO/IEC* 27001 визначає необхідність ведення журналів подій інформаційної безпеки, контролю доступу до ресурсів та здійснення постійного моніторингу функціонування ІС.

Рекомендації *National Institute of Standards and Technology (NIST)* також передбачають використання механізмів централізованого збору подій безпеки, аналізу журналів та виявлення аномальної активності користувачів і мережевих вузлів.

Національні нормативні документи України у сфері технічного захисту інформації, зокрема НД ТЗІ 2.5-004-99, визначають критерії оцінювання захищеності інформації від несанкціонованого доступу та встановлюють вимоги до реалізації механізмів контролю доступу і реєстрації подій безпеки. Крім того, НД ТЗІ 3.7-003-2005 регламентує порядок створення комплексних систем захисту інформації в ІКС, що передбачає впровадження засобів контролю подій інформаційної безпеки та моніторингу стану інформаційного середовища.

Таким чином, використання засобів моніторингу мережі є необхідною складовою виконання вимог нормативної бази інформаційної безпеки [5].

Для контролю стану мережевої інфраструктури використовуються різні програмні засоби моніторингу, які відрізняються функціональними можливостями, складністю впровадження та орієнтацією на конкретні задачі інформаційної безпеки.

Одним із найбільш поширених рішень є система *Zabbix*, яка належить до класу відкритого програмного забезпечення та використовується для контролю працездатності серверів, мережевих пристроїв і служб. Вона дозволяє здійснювати збір статистичних показників функціонування інфраструктури, аналіз навантаження та формування повідомлень про відмови обладнання. Основною особливістю системи є підтримка агентного та безагентного моніторингу мережевих вузлів.

Система *Nagios Core* також належить до відкритих рішень інфраструктурного моніторингу та забезпечує контроль серверів, мережевого обладнання, сервісів *DNS, FTP, SMTP* та інших компонентів ІС. Вона підтримує розширення функціональних можливостей за допомогою великої кількості додаткових модулів та плагінів, що дозволяє адаптувати систему до різних умов використання.

Серед комерційних систем моніторингу слід відзначити *PRTG Network Monitor*, який забезпечує централізований контроль стану мережі, серверів, сервісів і мережевого трафіку з використанням спеціалізованих сенсорів моніторингу. Система підтримує автоматичне виявлення мережевих пристроїв, формування звітів і повідомлень у режимі реального часу, а також створення візуальних карт мережевої інфраструктури.

Для реалізації функцій аналізу мережевої безпеки використовується платформа *Security Onion*, яка поєднує інструменти мережевого моніторингу безпеки та системи виявлення вторгнень. Вона базується на використанні таких компонентів, як *Suricata* та *Zeek*, що забезпечують аналіз мережевого трафіку на основі сигнатур та поведінкових правил [13, 14].

Порівняльна характеристика систем моніторінгу мережі наведена в таблиці 1.1

Таблиця 1.1 – Порівняльна характеристика систем моніторингу мережі

Критерій	***Zabbix***	***Nagios Core***	***PRTG Network Monitor***	***Security Onion***
Тип системи	Інфраструктурний моніторинг	Інфраструктурний моніторинг	Комплексний моніторинг мережі	Платформа *NSM/IDS*
Ліцензія	*Open Source*	*Open Source*	Комерційна (є *freeware*)	*Open Source*
Основне призначення	Контроль доступності вузлів, сервісів, метрик	Контроль стану *IT*-інфраструктури	Моніторинг мережі, сервісів, трафіку	Виявлення вторгнень та аналіз трафіку
Аналіз мережевого трафіку	Обмежений	Обмежений	Частковий	Повний
IDS-функції	Немає	Немає	Частково	Є
Підтримка *SNMP*	Так	Так	Так	Так
Виявлення аномалій	Частково	Частково	Так	Так
Автоматичне виявлення вузлів	Так	Частково	Так	Так
Вимоги до ресурсів	Середні	Невисокі	Середні	Високі
Складність налаштування	Середня	Висока	Невисока	Висока
Орієнтація на безпеку	Низька	Низька	Середня	Висока
Доцільність для малих мереж	Висока	Висока	Висока	Обмежена

Таким чином, існуючі системи моніторингу дозволяють здійснювати контроль стану інформаційної інфраструктури, проте більшість із них орієнтовані або на контроль працездатності обладнання, або на глибокий аналіз трафіку із значними ресурсними витратами [9 -12].

Незважаючи на значну функціональність сучасних систем моніторингу мережі, важливим фактором під час їх впровадження є економічна складова використання програмного забезпечення. Більшість професійних платформ мережевого моніторингу орієнтовані на корпоративний сегмент і потребують значних фінансових витрат на придбання ліцензій, підтримку серверної інфраструктури та технічне обслуговування.

Особливо актуальною проблема вартості є для навчальних закладів, невеликих організацій та локальних ІКС, де використання дорогих комерційних продуктів може бути економічно недоцільним. Крім того, частина систем має обмеження безкоштовних версій або потребує додаткових платних модулів для реалізації функцій контролю безпеки мережі.

Для оцінювання економічної доцільності використання існуючих систем моніторингу проведено порівняння їх орієнтовної вартості та особливостей ліцензування, результати якого наведено в таблиці 1.2 [36 - 38].

Як показано в таблиці 1.2, навіть системи моніторингу з відкритим програмним кодом часто потребують додаткових витрат на технічну підтримку, серверну інфраструктуру та адміністрування. Комерційні рішення, такі як *PRTG Network Monitor*, *Nagios XI* або *SolarWinds NPM*, характеризуються високою вартістю ліцензування та масштабування, що може бути економічно недоцільним для невеликих локальних ІКС, навчальних закладів або спеціалізованих мереж.

Крім фінансових витрат, значна частина існуючих систем має надлишкову функціональність та високі вимоги до апаратних ресурсів, що ускладнює їх використання у невеликих мережевих середовищах. У зв’язку з цим виникає доцільність створення власного програмного додатку контролю стану мережі, який забезпечуватиме реалізацію необхідних функцій моніторингу та виявлення несанкціонованих пристроїв при мінімальних витратах ресурсів і без необхідності придбання дорогих ліцензій.

Таблиця 1.2 — Порівняння вартості та особливостей ліцензування систем моніторингу мережі (2026 р.)

Система	**Тип ліцензії**	**Орієнтовна вартість у 2026 р.**	**Особливості ліцензування**	**Обмеження**
Zabbix	*Open Source* + платна підтримка	від 2500–5000 *USD*/рік за *Enterprise Support*	Безкоштовне використання, окремо оплачується технічна підтримка	Потребує складного адміністрування
Nagios Core	*Open Source*	Безкоштовно	Відкритий код	Обмежений інтерфейс та складне налаштування
Nagios XI	Комерційна	від 1995 *USD* за 100 вузлів + підтримка	*Perpetual License* + щорічне обслуговування	Платні модулі та розширення
PRTG Network Monitor	Комерційна	від 2400 *USD*/рік (500 *sensors*)	Підписка за кількістю сенсорів	Обмеження *free*-версії
Security Onion	*Open Source*	Безкоштовно	Відкритий код	Високі вимоги до серверної інфраструктури
SolarWinds NPM	Комерційна	від 3000–10000 *USD*/рік залежно від кількості вузлів	Підписка або *SaaS*-модель	Висока вартість масштабування

1.3 Обгрунтування необхідності створення власного програмного додатку

Практика експлуатації ІС показує, що в багатьох випадках виникає потреба саме у компактному програмному інструменті, здатному виконувати базові функції контролю мережевої активності без необхідності розгортання складної інфраструктури мережевого моніторингу. Такий інструмент повинен забезпечувати оперативне виявлення змін у структурі мережі, фіксацію появи нових вузлів, контроль доступності мережевих ресурсів та виявлення ознак підозрілої активності.

Особливо важливою вимогою до подібного програмного засобу є його автономність. У ряді випадків використання зовнішніх хмарних сервісів або централізованих систем збору подій інформаційної безпеки є обмеженим або

недоцільним з міркувань конфіденційності інформації, вимог нормативної документації або особливостей експлуатації ІС. Автономний програмний додаток дозволяє забезпечити локальний контроль мережевої активності без передачі службових даних до зовнішніх середовищ.

Використання легкого програмного рішення забезпечує можливість швидкого розгортання системи моніторингу без значних витрат часу та ресурсів, що є важливим фактором під час впровадження засобів захисту інформації у невеликих ІКС.

Необхідність розроблення власного програмного додатку моніторингу мережі обумовлюється також специфікою умов експлуатації ІС різного призначення.

У навчальних мережах характерною особливістю є велика кількість користувачів із різним рівнем підготовки у сфері інформаційних технологій. Часто в таких мережах виконуються лабораторні роботи, пов’язані з дослідженням мережевих протоколів, аналізом трафіку або використанням спеціалізованих утиліт адміністрування. У результаті цього підвищується ймовірність виникнення несанкціонованих підключень або змін конфігурації мережевих вузлів. Використання програмного додатку контролю стану мережі дозволяє оперативно фіксувати подібні зміни та своєчасно реагувати на потенційно небезпечні події.

У військових ІКС особливе значення має забезпечення автономності функціонування засобів захисту інформації. Використання зовнішніх сервісів моніторингу або централізованих систем обробки подій може бути обмежене вимогами нормативних документів або умовами експлуатації. У таких випадках локальний програмний додаток контролю мережевої активності дозволяє забезпечити необхідний рівень спостереження за станом мережі без залучення зовнішніх ресурсів.

Для корпоративних мереж актуальною є задача контролю внутрішніх мережевих взаємодій між користувачами та сервісами ІС. Значна частина сучасних кіберінцидентів пов’язана саме з внутрішніми загрозами або

використанням скомпрометованих облікових записів. Застосування програмного додатку моніторингу дозволяє своєчасно виявляти аномальні зміни мережевої активності та підвищувати ефективність реагування на інциденти інформаційної безпеки.

Отже, універсальність використання програмного додатку контролю стану мережі в різних типах ІС підтверджує доцільність його розроблення.

Розроблення власного програмного додатку контролю стану мережі має ряд суттєвих переваг порівняно з використанням готових програмних комплексів моніторингу.

Однією з основних переваг є можливість адаптації програмного забезпечення до конкретних умов функціонування ІС. На відміну від універсальних програмних продуктів, власний додаток може бути оптимізований відповідно до топології мережі, структури інформаційних ресурсів та вимог адміністратора безпеки.

Важливою перевагою є також відкритість програмного коду, що забезпечує можливість перевірки реалізованих алгоритмів обробки мережевих подій та виключає наявність прихованих механізмів збору інформації. Це особливо актуально для інформаційних систем із підвищеними вимогами до рівня довіри програмного забезпечення.

Використання власного програмного рішення дозволяє реалізувати лише ті функції моніторингу, які є необхідними для конкретної ІС. Це забезпечує зменшення навантаження на обчислювальні ресурси та спрощує процес експлуатації програмного забезпечення.

Ще однією перевагою є можливість подальшого розвитку програмного додатку відповідно до змін умов експлуатації ІС та появи нових кіберзагроз. Гнучкість архітектури програмного забезпечення дозволяє поступово розширювати його функціональні можливості без необхідності повної заміни програмного комплексу [8].

Висновок до першого розділу

У результаті проведеного аналізу сучасних мережевих кіберзагроз встановлено, що більшість атак на ІС реалізується через мережеву інфраструктуру шляхом використання механізмів віддаленого доступу, компрометації облікових записів та експлуатації вразливостей мережевих сервісів. Зростання кількості кіберінцидентів та підвищення рівня їх складності обумовлюють необхідність постійного контролю стану мережі.

Проведений аналіз існуючих систем моніторингу показав, що більшість із них орієнтовані на контроль працездатності обладнання або потребують значних ресурсів для впровадження. Водночас вимоги міжнародних стандартів *ISO/IEC* 27001, рекомендацій *NIST* та НД ТЗІ України передбачають реалізацію механізмів реєстрації подій безпеки та контролю мережевої активності.

Таким чином, створення власного програмного додатку контролю стану мережі дозволяє забезпечити необхідний рівень ефективності моніторингу мережевої активності при оптимальному використанні ресурсів інформаційної системи та високому рівні адаптації до конкретних умов її функціонування.

РОЗДІЛ 2. МЕТОДИ ТА ТЕХНОЛОГІЇ ВИЯВЛЕННЯ НЕСАНКЦІОНОВАНОГО ДОСТУПУ І КОНТРОЛЮ СТАНУ МЕРЕЖІ

2.1. Методи збору та аналізу мережевих даних

Ефективність виявлення несанкціонованого доступу до ІКС значною мірою залежить від можливості своєчасного отримання та аналізу інформації про стан мережевої інфраструктури. Для цього використовуються спеціалізовані методи збору мережевих даних [1, 6], які дозволяють визначати структуру мережі, активність вузлів, параметри передавання трафіку та ознаки аномальної поведінки користувачів або пристроїв.

У сучасних системах контролю стану мережі застосовуються два основні підходи до отримання інформації про мережеву активність:

- пасивний моніторинг мережі;
- активне сканування мережевого середовища.

Пасивні методи передбачають аналіз наявної службової інформації мережі без генерації додаткового трафіку, тоді як активні методи базуються на формуванні спеціальних запитів до мережевих вузлів з метою отримання інформації про їх стан.

Комплексне використання зазначених методів дозволяє підвищити ефективність виявлення ознак несанкціонованого доступу та забезпечити своєчасне реагування на потенційні загрози інформаційній безпеці.

Пасивний моніторинг мережі є одним із найбільш ефективних способів отримання інформації про стан мережевої інфраструктури без створення додаткового навантаження на канали зв’язку [6]. Основною особливістю такого підходу є використання вже наявних службових даних мережевих протоколів та журналів подій мережевого обладнання.

До основних джерел інформації під час пасивного моніторингу (рис.2.1) належать:

- *ARP*-таблиці мережевих пристроїв;

- журнали *DHCP*-серверів;

- мережевий трафік локального сегменту.

Рис. 2.1 — Основні джерела даних пасивного моніторингу мережі

Аналіз зазначених джерел дозволяє визначати структуру мережі, ідентифікувати активні вузли та виявляти появу нових або підозрілих пристроїв.

1. Використання *ARP*-таблиць для моніторингу мережі

ARP-таблиця містить відповідності між *IP*-адресами та фізичними *MAC*-адресами мережевих пристроїв [17]. Аналіз таких таблиць дозволяє виявляти активні вузли локальної мережі, контролювати зміну мережевих параметрів та визначати появу нових або невідомих пристроїв. Приклад *ARP*-таблиці локального сегменту мережі наведено в таблиці 2.1.

Таблиця 2.1 — Приклад *ARP*-таблиці локальної мережі

***IP*-адреса**	***MAC*-адреса**	**Тип запису**	**Опис пристрою**
192.168.1.1	3C:52:82:4F:11:20	Динамічний	Маршрутизатор
192.168.1.10	A8:5E:45:2B:91:7C	Динамічний	Робоча станція
192.168.1.15	90:2B:34:AA:18:6D	Динамічний	Ноутбук користувача
192.168.1.20	44:D9:E7:71:2C:11	Динамічний	Мережевий принтер
192.168.1.35	00:25:96:FF:4A:8B	Динамічний	Невідомий пристрій
192.168.1.50	B4:2E:99:1D:CC:21	Статичний	Сервер локальної мережі

Як показано в таблиці 2.2, більшість записів *ARP*-таблиці відповідають відомим пристроям локальної мережі. Разом із тим вузол із *IP*-адресою 192.168.1.35 має невідому *MAC*-адресу та не входить до переліку дозволених пристроїв, що може свідчити про наявність несанкціонованого підключення або спробу прихованого доступу до мережі. Аналіз подібних змін дозволяє своєчасно виявляти аномальну мережеву активність у ІКС [19].

Контроль *ARP*-таблиць є ефективним методом виявлення:

- появи нових пристроїв у мережі;
- підміни мережевих адрес;
- атак типу *ARP-spoofing*;
- несанкціонованого підключення користувачів.

Регулярний аналіз *ARP*-таблиць дозволяє своєчасно виявляти аномальні зміни структури локальної мережі та підвищує ефективність контролю доступу до інформаційних ресурсів.

2. Аналіз журналів *DHCP*-серверів

DHCP-сервер забезпечує автоматичну видачу *IP*-адрес мережевим пристроям [18], тому його журнали подій містять важливу інформацію про підключення клієнтів до мережі.

Аналіз *DHCP*-логів дозволяє:

- визначати час підключення пристроїв;
- ідентифікувати *MAC*-адреси клієнтів;
- відстежувати зміну *IP*-адрес;
- виявляти несанкціоновані підключення до мережі.

Використання *DHCP*-логів є особливо ефективним у корпоративних та навчальних мережах із великою кількістю користувачів.

3. Прослуховування мережевого сегменту

Прослуховування мережевого сегменту передбачає аналіз мережевих пакетів, що передаються локальною мережею [19]. Для цього використовуються спеціалізовані програмні засоби аналізу трафіку.

Застосування такого підходу дозволяє:

- контролювати активність мережевих вузлів;

- виявляти підозрілу мережеву взаємодію;

- аналізувати структуру мережевого трафіку;

- визначати ознаки сканування портів;

- виявляти спроби несанкціонованого доступу.

Таким чином, пасивний моніторинг мережі забезпечує ефективний контроль стану мережевої інфраструктури без створення додаткового мережевого навантаження.

Активні методи збору мережевих даних базуються на формуванні спеціальних запитів до мережевих вузлів з метою отримання інформації про їх доступність, конфігурацію та відкриті мережеві служби.

Основними перевагами активного сканування є:

- швидке визначення активних вузлів мережі;

- можливість отримання детальної інформації про мережеві сервіси;

- виявлення відкритих портів;

- контроль доступності мережевих ресурсів.

До найбільш поширених методів активного сканування належать *ping-sweep*, *ARP-scan* та порт-сканування (рис. 2.2).

Рис. 2.2 — Основні методи активного сканування мережі

1. Метод *ping-sweep*

Ping-sweep використовується для визначення активних вузлів мережі шляхом надсилання *ICMP*-запитів до заданого діапазону *IP*-адрес [20].

Цей метод дозволяє:

- визначати доступні пристрої;

- перевіряти працездатність вузлів;

- контролювати стан мережевого сегменту;

- виявляти нові підключення до мережі.

Ping-sweep широко використовується адміністраторами мережі для швидкої інвентаризації мережевої інфраструктури.

2. Метод *ARP-scan*

ARP-scan застосовується для визначення активних пристроїв локальної мережі на канальному рівні моделі *OSI* [21].

На відміну від *ICMP*-запитів, *ARP*-запити не блокуються більшістю мережевих екранів локального сегменту, що підвищує ефективність цього методу.

Використання *ARP-scan* дозволяє:

- отримувати *MAC*-адреси пристроїв;

- визначати активні вузли локальної мережі;

- виявляти несанкціоновані підключення;

- контролювати зміну топології мережі.

3. Метод порт-сканування

Порт-сканування використовується для визначення відкритих мережевих служб на вузлах інформаційної системи [22].

Застосування цього методу дозволяє:

- визначати активні мережеві сервіси;

- контролювати конфігурацію вузлів;

- виявляти потенційні вразливості;

- визначати ознаки несанкціонованої активності.

Порт-сканування є одним із найбільш поширених інструментів як мережевих адміністраторів, так і систем виявлення вторгнень.

Комплексне використання пасивних та активних методів збору мережевих даних відповідає рекомендаціям сучасних систем виявлення вторгнень та забезпечує підвищення ефективності контролю стану ІКС.

2.2. Технології ідентифікації та детектування несанкціонованих пристроїв

Одним із ключових завдань забезпечення безпеки ІКС є своєчасне виявлення несанкціонованих пристроїв, що підключаються до локальної мережі. Такі пристрої можуть використовуватися для проведення мережевої розвідки, перехоплення трафіку, підміни адрес або реалізації атак на мережеву інфраструктуру [6].

Сучасні методи ідентифікації мережевих вузлів базуються на аналізі параметрів канального рівня моделі *OSI*, дослідженні таблиць відповідностей мережевих адрес, контролі змін топології мережі та застосуванні евристичних алгоритмів визначення аномальної активності.

Застосування зазначених підходів дозволяє підвищити ефективність виявлення ознак несанкціонованого доступу до ресурсів ІКС.

Одним із найбільш поширених способів порушення безпеки локальних мереж є атака типу *ARP-spoofing*, яка базується на підміні відповідностей між *IP*-адресами та *MAC*-адресами мережевих пристроїв [17]. У результаті реалізації такої атаки зловмисник отримує можливість перехоплювати мережевий трафік або змінювати маршрути передавання даних у межах локального сегменту мережі.

Принцип реалізації *ARP-spoofing* (рис. 2.3) полягає у підміні відповідностей між *IP*-адресами та *MAC*-адресами мережевих вузлів локальної мережі. У результаті цього мережевий трафік може бути перенаправлений через пристрій зловмисника, що дозволяє виконувати перехоплення переданих даних. [22].

Як показано на рис. 2.3, зловмисник формує підроблені *ARP*-відповіді, унаслідок чого вузли локальної мережі починають передавати мережевий трафік через пристрій атакуючого. Це дозволяє реалізувати атаку типу *MITM* та здійснювати перехоплення або модифікацію даних.

Рис. 2.3 — Схема реалізації атаки типу *ARP-spoofing* у локальній мережі

До основних ознак *ARP-spoofing* належать:

- поява дубльованих *MAC*-адрес у мережевому сегменті;
- часті зміни відповідностей *IP*-адрес;
- наявність конфліктів *ARP*-записів;
- різке збільшення кількості *ARP*-повідомлень у мережі.

Для детектування *ARP-spoofing* використовуються такі методи контролю *ARP*-таблиць та аналізу мережевих пакетів [19]:

- аналіз змін *ARP*-таблиць;
- порівняння відповідностей *MAC*-адрес;
- контроль динаміки *ARP*-запитів;
- використання списків довірених пристроїв;

- аналіз часових інтервалів оновлення *ARP*-записів.

Застосування зазначених методів дозволяє своєчасно виявляти ознаки підміни мережевих адрес та запобігати несанкціонованому доступу до інформаційних ресурсів ІКС.

Прихованими або «сірими» пристроями називаються мережеві вузли, які не зареєстровані у централізованих системах адміністрування мережі або не входять до переліку дозволених пристроїв ІКС.

Поява таких пристроїв може бути пов’язана з:

- підключенням несанкціонованого обладнання;

- використанням особистих пристроїв користувачів;

- діяльністю зловмисників;

- помилками конфігурації мережі.

Виявлення «сірих» пристроїв здійснюється шляхом аналізу журналів підключень та таблиць відповідностей мережевих адрес (рис. 2.4) [23]:

- аналізу *DHCP*-журналів;

- порівняння *MAC*-адрес із базою дозволених вузлів;

- аналізу *ARP*-таблиць;

- моніторингу появи нових *IP*-адрес;

- контролю змін топології мережі.

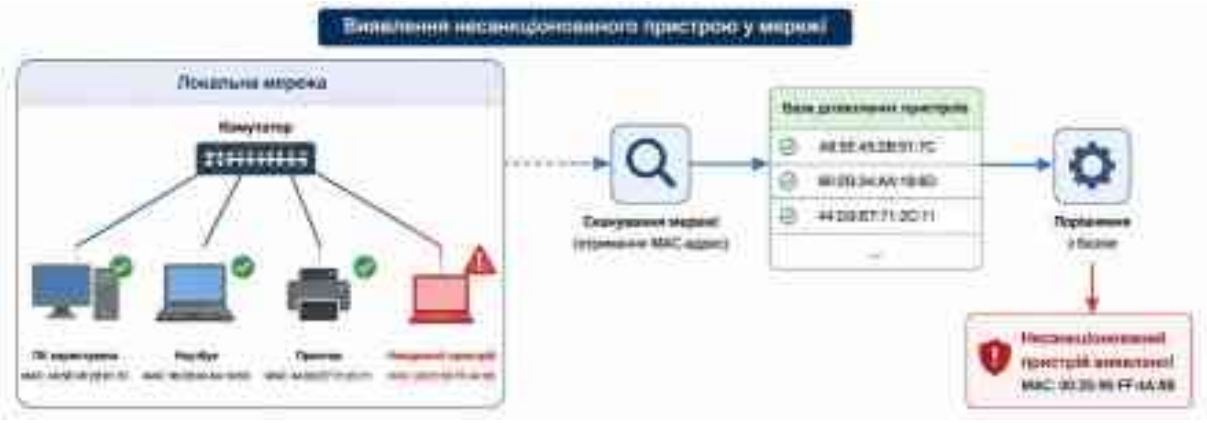

Рис. 2.4 — Схема виявлення несанкціонованого пристрою у локальному сегменті мережі

Застосування зазначених методів дозволяє своєчасно виявляти несанкціоновані підключення до мережі та забезпечувати контроль доступу до ресурсів ІКС.

MAC-адреса є унікальним ідентифікатором мережевого інтерфейсу пристрою, що використовується на канальному рівні моделі *OSI* для організації передавання даних у локальній мережі. Аналіз *MAC*-адрес дозволяє виконувати ідентифікацію мережевих вузлів та контролювати зміни їх конфігурації.

До основних методів швидкої ідентифікації *MAC*-адрес належать:

- аналіз *ARP*-таблиць;
- використання *DHCP*-логів;
- зіставлення *MAC*-адрес із базою виробників обладнання;
- контроль змін відповідностей *IP*- та *MAC*-адрес;
- моніторинг повторного використання *MAC*-адрес.

Зміна *MAC*-адреси може використовуватися зловмисниками для приховування своєї присутності у мережі або обходу механізмів контролю доступу. Тому контроль відповідностей мережевих адрес є важливим елементом забезпечення безпеки ІКС (рис. 2.5).

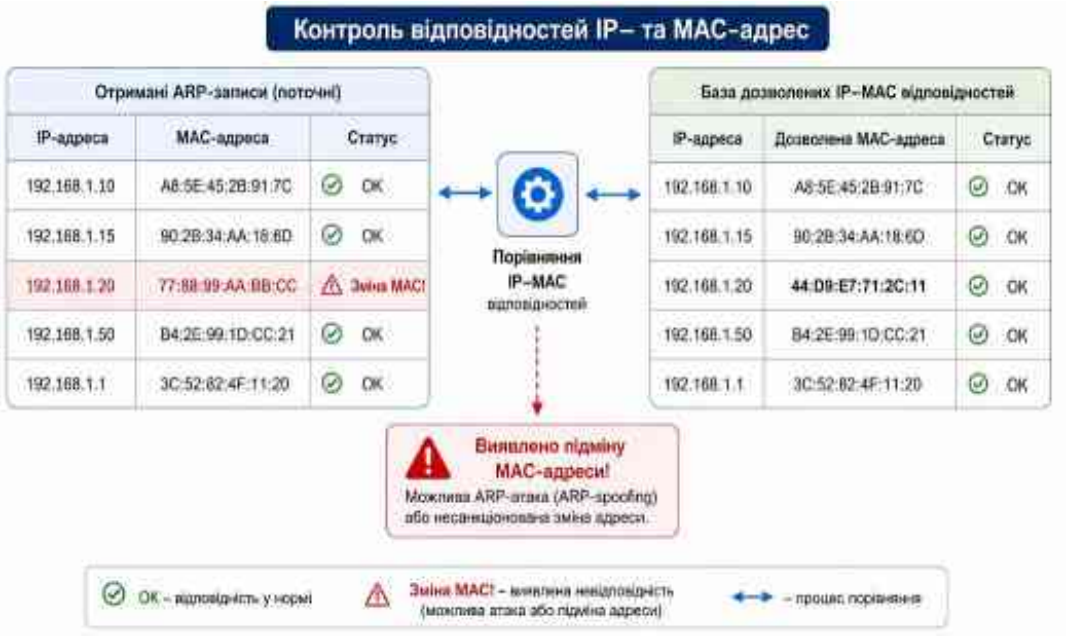

Рис. 2.5 — Контроль відповідностей *IP*- та *MAC*-адрес для виявлення аномалій

Особливо ефективним є використання автоматизованих засобів аналізу таблиць відповідностей мережевих адрес, що дозволяє оперативно виявляти аномальні зміни параметрів мережевих вузлів.

Сучасні системи контролю стану мережі використовують різні підходи до визначення аномальної активності мережевих вузлів. До найбільш ефективних належать порогові та евристичні моделі аналізу мережевих параметрів.

Порогові методи базуються на встановленні допустимих значень параметрів функціонування мережі. У разі перевищення встановлених порогових значень система формує повідомлення про можливу наявність загрози інформаційній безпеці.

До параметрів контролю можуть належати:

- кількість *ARP*-запитів;
- кількість активних вузлів мережі;
- частота зміни *MAC*-адрес;
- інтенсивність мережевого трафіку;
- кількість спроб підключення до вузлів мережі.

Евристичні методи аналізу дозволяють визначати аномалії на основі поведінкових характеристик мережевих пристроїв [24]. Використання таких методів забезпечує можливість виявлення нових типів атак, які не мають відомих сигнатур.

Комплексне застосування порогових та евристичних моделей аналізу відповідає сучасним підходам до побудови систем виявлення вторгнень, дозволяє підвищити ефективність виявлення несанкціонованих пристроїв у мережі та забезпечити належний рівень захисту ІКС [6, 25].

Різні методи детектування несанкціонованих пристроїв у мережі мають відмінні характеристики ефективності, швидкості роботи та області застосування. Для визначення доцільності використання кожного з методів проведено їх порівняльний аналіз, результати якого наведено в таблиці 2.2.

Таблиця 2.2 — Порівняння методів детектування несанкціонованих пристроїв у локальній мережі

Метод детектування	**Рівень *OSI***	**Принцип роботи**	**Переваги**	**Недоліки**	**Доцільність використання**
Аналіз *ARP*-таблиць	Канальний	Порівняння відповідностей *IP–MAC*	Простота реалізації, швидкість	Не працює поза локальним сегментом	Локальні мережі
Аналіз *DHCP*-журналів	Мережевий	Контроль видачі *IP*-адрес	Дозволяє визначити нові підключення	Працює лише при використанні *DHCP*	Корпоративні мережі
ARP-scan	Канальний	Активний запит до вузлів мережі	Висока точність виявлення	Створює додатковий трафік	Інвентаризація мережі
Ping-sweep	Мережевий	*ICMP*-запити до вузлів	Простота використання	Може блокуватися *firewall*	Первинний моніторинг
Порт-сканування	Транспорт-ний	Аналіз відкритих портів	Детальна інформація про вузли	Потребує більше часу	Аудит безпеки
Контроль *MAC*-адрес (*OUI*)	Канальний	Аналіз виробника пристрою	Виявлення підроблених пристроїв	Можлива підміна *MAC*	Контроль доступу
Порогові методи	Різні	Контроль перевищення значень параметрів	Простота реалізації	Низька адаптивність	Реальний час
Евристичні методи	Різні	Аналіз поведінкових характеристик	Виявлення нових атак	Складність реалізації	*IDS*-системи

2.3. Вимоги до функціоналу, безпеки та продуктивності майбутнього додатку

Розроблення програмного додатку контролю стану мережі та виявлення несанкціонованого доступу до ІКС потребує формування чітких функціональних та нефункціональних вимог відповідно до стандартів якості програмного забезпечення [26]. Формалізація таких вимог дозволяє забезпечити відповідність програмного засобу сучасним вимогам інформаційної безпеки, підвищити ефективність його використання та забезпечити можливість подальшого розвитку програмного продукту.

Визначення вимог до програмного додатку здійснюється з урахуванням особливостей функціонування локальних мереж, необхідності оперативного виявлення змін структури мережі, контролю підключення нових пристроїв та аналізу параметрів мережевої активності користувачів.

Основні вимоги до програмного додатку поділяються на:

- функціональні;

- нефункціональні.

На рис. 2.6 представлена узагальнена схема вимог до програмного додатку контролю стану мережі.

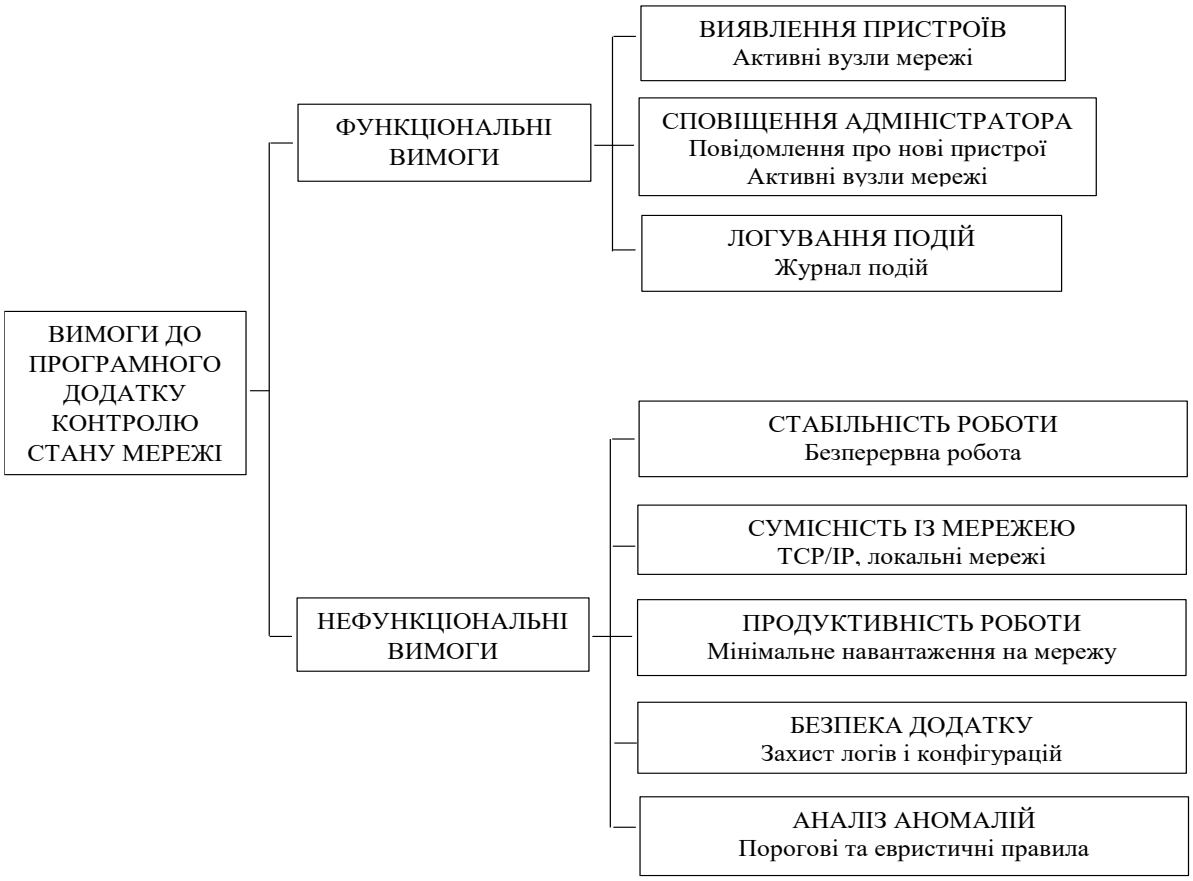

Рис. 2.6 — Узагальнена схема функціональних і нефункціональних вимог до програмного додатку контролю стану мережі

Функціональні вимоги визначають перелік основних можливостей програмного додатку, необхідних для реалізації контролю стану мережі та виявлення несанкціонованих пристроїв у ІКС.

До основних функціональних вимог належать:

1. Виявлення мережевих пристроїв

Програмний додаток повинен забезпечувати автоматичне визначення активних вузлів локальної мережі шляхом аналізу таблиць відповідностей мережевих адрес та використання методів активного сканування мережевого сегменту, що є базовою функцією систем мережевого моніторингу [6]. Реалізація цієї функції дозволяє формувати актуальний перелік підключених пристроїв та контролювати зміни топології мережі.

Особливу увагу необхідно приділяти виявленню нових або невідомих пристроїв, що можуть свідчити про наявність ознак несанкціонованого доступу до ресурсів ІКС.

2. Формування повідомлень про події безпеки

Програмний додаток повинен забезпечувати автоматичне формування повідомлень у разі виявлення підозрілої мережевої активності або змін структури локального сегменту мережі, відповідає рекомендаціям систем виявлення вторгнень *NIST* [6].

До подій, що підлягають реєстрації, належать:

- поява нового мережевого пристрою;
- зміна *MAC*-адреси вузла;
- перевищення порогових значень мережевої активності;
- конфлікти мережевих адрес;
- ознаки *ARP-spoofing*.

Оперативне інформування адміністратора дозволяє скоротити час реагування на інциденти інформаційної безпеки.

3. Логування подій мережевої активності

Важливою функціональною вимогою є забезпечення можливості ведення журналу подій мережевої активності відповідно до рекомендацій управління журналами безпеки [23].

Програмний додаток повинен забезпечувати:

- реєстрацію подій підключення пристроїв;

- фіксацію змін мережевих параметрів;

- збереження історії мережевої активності;

- формування звітів про стан мережі.

Використання механізмів журналювання дозволяє здійснювати подальший аналіз інцидентів інформаційної безпеки та підвищує ефективність контролю функціонування ІКС.

Нефункціональні вимоги визначають якісні характеристики програмного додатку та забезпечують ефективність його функціонування в умовах експлуатації ІКС.

До основних нефункціональних вимог належать стабільність функціонування, сумісність із мережевим середовищем, продуктивність роботи та відповідність вимогам інформаційної безпеки.

1. Стабільність функціонування

Програмний додаток повинен забезпечувати безперервний моніторинг стану мережі протягом тривалого часу без виникнення критичних помилок або збоїв.

Стабільність функціонування забезпечується:

- використанням оптимізованих алгоритмів обробки мережевих даних відповідно до вимог якості програмного забезпечення [26];

- контролем використання системних ресурсів;

- реалізацією механізмів обробки помилок.

2. Сумісність із мережевим середовищем

Програмний додаток повинен забезпечувати коректну роботу у стандартних локальних мережах із використанням протоколів *TCP/IP*.

Сумісність програмного забезпечення передбачає можливість його використання:

- у навчальних мережах;

- у корпоративних ІКС;

- у сегментованих локальних мережах.

3. Продуктивність роботи

Програмний додаток повинен забезпечувати оперативне отримання інформації про стан мережі без перевантаження мережевої інфраструктури відповідно до рекомендацій побудови систем мережевого моніторингу [27].

Основними вимогами до продуктивності є:

- мінімізація затримок під час аналізу мережевих даних;
- оптимальне використання процесорних ресурсів;
- обмеження обсягів службового мережевого трафіку.

4. Забезпечення інформаційної безпеки програмного додатку

Програмний додаток повинен відповідати вимогам безпеки програмного забезпечення згідно рекомендацій *OWASP* щодо безпечної розробки програмних систем [28] та забезпечувати захист службових даних, що використовуються під час моніторингу мережевої активності.

До основних вимог безпеки належать:

- контроль доступу до журналів подій;
- захист конфігураційних файлів;
- обмеження прав користувачів;
- забезпечення цілісності збережених даних.

Таким чином, сформульовані функціональні та нефункціональні вимоги відповідають сучасним підходам до розроблення захищених програмних засобів контролю стану ІКС [6, 29].

Висновок до другого розділу

У другому розділі кваліфікаційної роботи було проаналізовано сучасні методи збору та аналізу мережевих даних, а також технології ідентифікації та детектування несанкціонованих пристроїв у ІКС. Розглянуто можливості використання пасивного моніторингу мережі на основі аналізу *ARP*-таблиць, *DHCP*-журналів та мережевого трафіку, а також методів активного сканування, зокрема *ping-sweep*, *ARP-scan* і порт-сканування.

Окрему увагу приділено технологіям виявлення атак типу *ARP-spoofing*, методам ідентифікації прихованих мережевих пристроїв, контролю

відповідностей *IP*- та *MAC*-адрес і застосуванню порогових та евристичних моделей визначення аномалій мережевої активності. Проведений аналіз підтвердив ефективність використання комплексного підходу до контролю стану мережі для своєчасного виявлення ознак несанкціонованого доступу.

Крім того, було сформульовано функціональні та нефункціональні вимоги до програмного додатку контролю стану мережі, що враховують особливості його використання в ІКС та забезпечують необхідний рівень стабільності, продуктивності та інформаційної безпеки.

Отримані результати створюють теоретичну основу для подальшого проєктування архітектури програмного додатку контролю стану мережі та реалізації механізмів виявлення несанкціонованих підключень у локальних мережах.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ДОДАТКУ КОНТРОЛЮ СТАНУ МЕРЕЖІ ТА ВИЯВЛЕННЯ НЕСАНКЦІОНОВАНОГО ДОСТУПУ ДО НЕЇ

3.1. Розроблення структури та алгоритму програмного додатку

Функціональна структура програмного додатку визначає загальну організацію системи контролю стану локальної мережі та виявлення потенційно несанкціонованого доступу. Її розроблення є важливим етапом проєктування, оскільки саме на цьому рівні формується логіка взаємодії окремих компонентів системи, визначаються інформаційні потоки та принципи обробки мережевих подій.

Основним призначенням програмного додатку є автоматизований контроль локального мережевого середовища, виявлення активних вузлів, аналіз змін мережевих параметрів, фіксація підозрілих подій та надання адміністратору інформації для прийняття рішень щодо безпеки мережі.

З огляду на поставлені функціональні вимоги, програмний додаток доцільно представити як модульну систему, що складається з взаємопов’язаних функціональних компонентів. Такий підхід спрощує логічну організацію системи, забезпечує чітке розмежування функцій та створює передумови для подальшого розвитку або модифікації окремих компонентів.

Функціональна структура програмного додатку наведена на рис. 3.1.

У структурі програмного додатку можна виділити чотири основні функціональні модулі: модуль збору мережевих даних, модуль аналізу та виявлення потенційно несанкціонованих пристроїв, модуль журналювання та формування сповіщень, а також модуль взаємодії з користувачем.

Модуль збору мережевих даних забезпечує отримання актуальної інформації про активні вузли локального сегмента мережі. Основною його

функцією є виявлення пристроїв, визначення їхніх *IP*- та *MAC*-адрес, а також формування первинного набору даних для подальшого аналізу. [1]

Модуль аналізу та виявлення потенційно несанкціонованих пристроїв здійснює логічну обробку отриманих мережевих даних. [6]. Основним завданням модуля є порівняння виявлених пристроїв із переліком раніше зафіксованих вузлів, перевірка відповідності *MAC*-ідентифікаторів, виявлення нових або підозрілих записів та аналіз аномальних змін мережевого середовища. Для кожного пристрою формується відповідний статус: *allowed*, *blocked* або *unknown*.

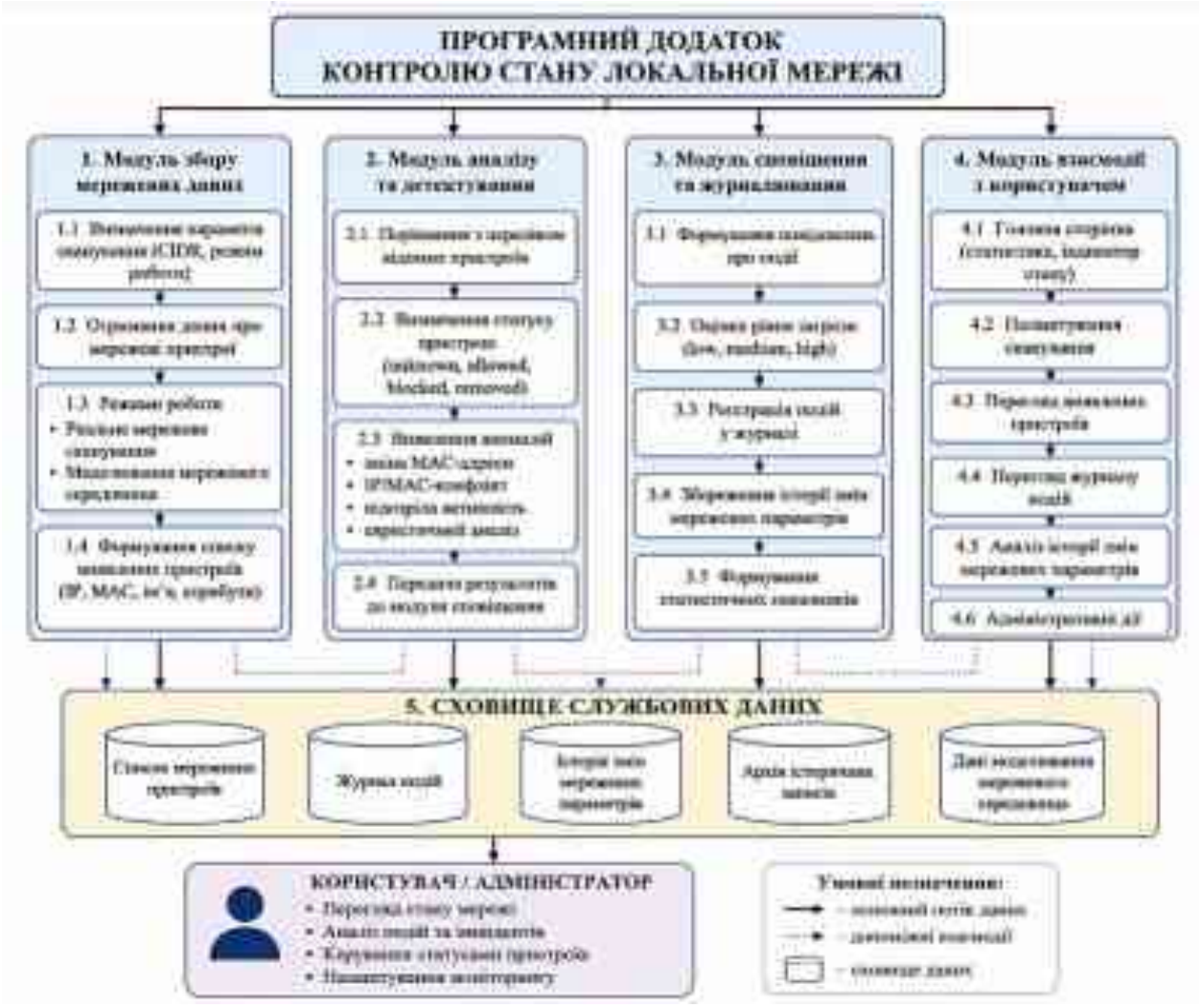

Рис. 3.1 — Функціональна структура програмного додатку

Модуль журналювання та формування сповіщень забезпечує реєстрацію подій, накопичення історичних даних про стан мережевого середовища та

формування повідомлень про виявлені інциденти або аномальні ситуації [23]. До журналу заносяться результати сканування, виявлення нових пристроїв, зміни *MAC*-адрес, потенційні конфлікти мережевих параметрів і службові події. Для зручності контролю реалізовано механізм оцінювання поточного рівня загроз на основі накопичених подій.

Модуль взаємодії з користувачем забезпечує відображення результатів моніторингу, статистичних показників, журналів подій та службової інформації, необхідної для контролю стану мережі та підтримки прийняття адміністративних рішень.

Взаємодія між зазначеними модулями реалізується за принципом послідовної обробки інформації: отримані мережеві дані передаються до аналітичного блоку, результати аналізу надходять до підсистеми журналювання, після чого інформація стає доступною для користувача у зручній формі.

Запропонована функціональна структура забезпечує логічну цілісність програмного додатку, підтримує розподіл функціонального навантаження між окремими компонентами та створює основу для реалізації алгоритмів моніторингу локального мережевого середовища.

Після визначення загальної структури програмного додатку доцільно розглянути алгоритм функціонування модуля збору мережевих даних (рис. 3.2), який є початковим етапом функціонування всієї системи моніторингу, оскільки саме на цьому етапі формується первинна інформаційна база для подальшого аналізу стану мережі. Від коректності та повноти зібраних даних залежить точність виявлення нових, підозрілих або потенційно несанкціонованих мережевих пристроїв. [6, 27].

Процес збору мережевої інформації передбачає послідовне виконання кількох логічних етапів: визначення параметрів сканування, вибір режиму роботи системи, отримання відомостей про активні мережеві вузли та формування структурованого переліку виявлених пристроїв.

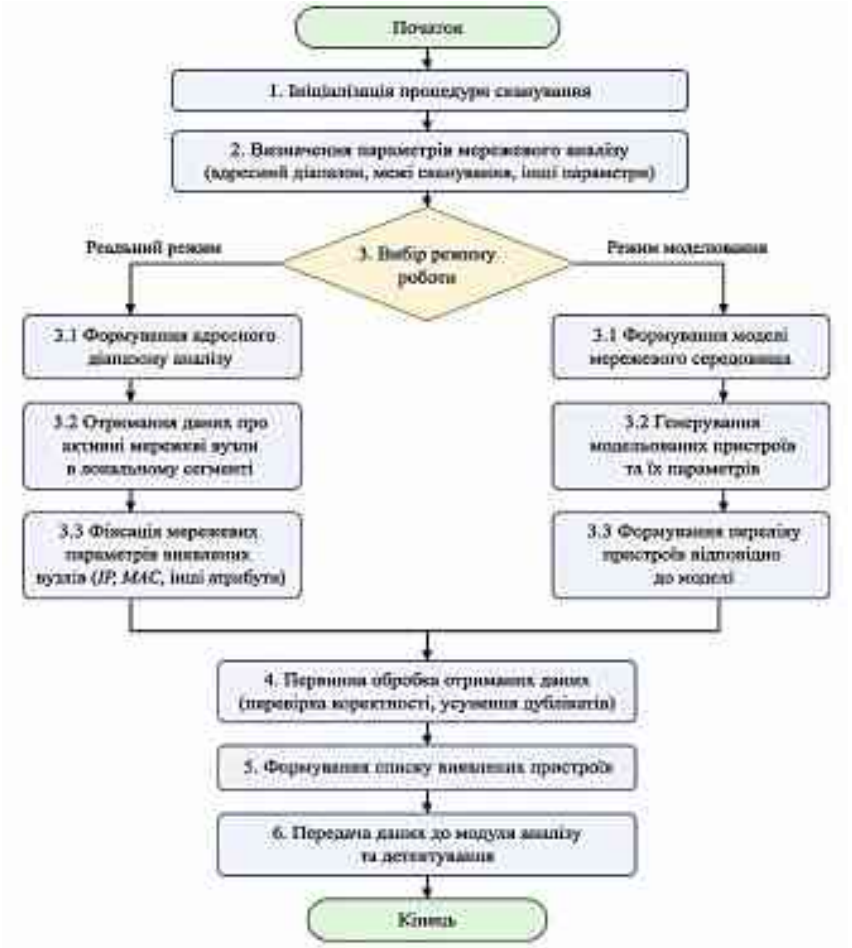

Рис. 3.2 — Алгоритм сканування мережі

На першому етапі визначаються параметри аналізованого локального мережевого сегмента, зокрема адресний діапазон, у межах якого виконуватиметься пошук активних вузлів. Це дозволяє обмежити область моніторингу конкретним сегментом мережі та підвищити ефективність аналізу.

Наступним кроком є вибір режиму роботи системи. Залежно від умов використання програмний додаток може функціонувати у режимі аналізу реального локального мережевого середовища або у режимі моделювання [45]. Перший варіант передбачає отримання фактичних даних із локального мережевого сегмента, тоді як другий використовується для тестування алгоритмів без потреби у зміні реального мережевого середовища. Після цього система виконує збір інформації про активні мережеві пристрої. Для кожного виявленого вузла фіксуються його основні мережеві характеристики,

необхідні для подальшої ідентифікації, аналізу та контролю стану мережевого середовища.

Завершальним етапом є формування структурованого списку виявлених мережевих пристроїв, який передається до модуля аналізу та детектування для подальшої логічної обробки.

Запропонований алгоритм забезпечує систематизоване отримання інформації про стан локального мережевого середовища та створює основу для подальшого аналізу потенційно підозрілих мережевих подій [6].

Після завершення етапу збору мережевих даних система переходить до аналізу отриманої інформації з метою виявлення потенційно несанкціонованих або підозрілих мережевих пристроїв. Саме цей етап є ключовим для функціонування програмного додатку, оскільки дозволяє не лише фіксувати наявність активних вузлів у мережі, але й оцінювати їхню допустимість та потенційний рівень загрози.

Основним завданням модуля аналізу є порівняння інформації про виявлені мережеві пристрої з уже відомими системі записами, визначення їхнього поточного статусу, виявлення змін мережевих параметрів та фіксація можливих ознак аномальної активності.

На початковому етапі аналізу для кожного виявленого мережевого вузла виконується перевірка наявності відповідного запису у внутрішньому переліку відомих пристроїв. Якщо пристрій уже був раніше зафіксований системою, здійснюється перевірка відповідності його мережевих характеристик попередньо збереженим параметрам.

Якщо пристрій відсутній у переліку відомих вузлів, він розглядається як новий або невідомий об’єкт мережевого середовища, що потребує подальшої оцінки з боку адміністратора.

У разі виявлення відомого пристрою система аналізує сталість його мережевих параметрів. Зміна відповідності між *IP*- та *MAC*-ідентифікаторами, поява конфліктних параметрів або інші нетипові зміни можуть свідчити про аномальну ситуацію або потенційну спробу несанкціонованого втручання

Для спрощення логіки аналізу доцільно використовувати систему адміністративних статусів, що відображає поточний стан кожного мережевого вузла:

unknown — новий або невідомий пристрій;

allowed — довірений пристрій;

blocked — пристрій, що розглядається як потенційно небажаний або обмежений;

removed — пристрій, який був виключений із активного переліку.

Такий підхід забезпечує формалізацію процесу прийняття рішень і спрощує подальший контроль стану мережевого середовища (рис. 3.3).

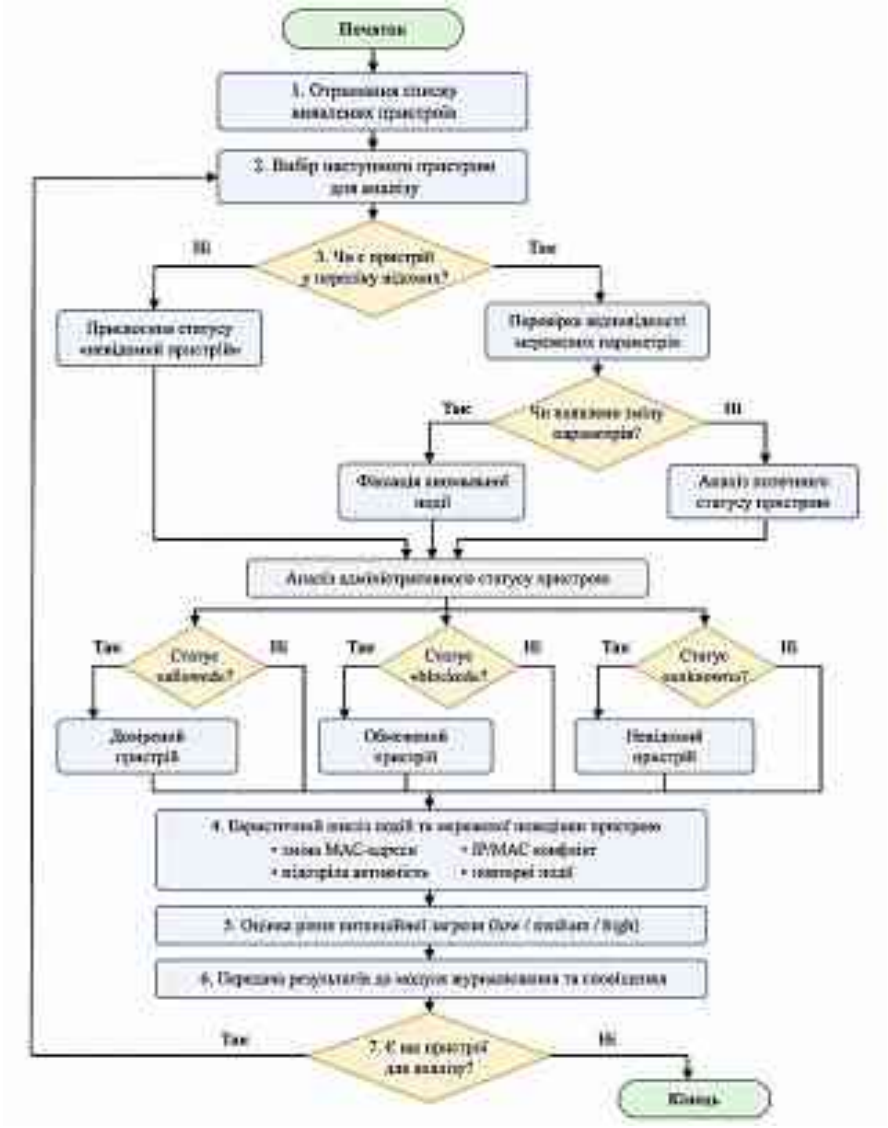

Рис. 3.3 – Алгоритм аналізу та детектування потенційно несанкціонованих пристроїв

Крім базової перевірки відомих вузлів, алгоритм передбачає аналіз потенційних аномалій [45]. До таких подій можуть належати нетипова зміна мережевих параметрів, повторна поява раніше невідомих пристроїв або інші відхилення від очікуваної структури локального сегмента.

Результатом роботи алгоритму є формування висновку щодо поточного стану кожного виявленого вузла та передача інформації до модуля журналювання й сповіщення для подальшої обробки.

Запропонований алгоритм забезпечує структурований підхід до оцінювання стану мережевого середовища та створює основу для своєчасного виявлення потенційно небезпечних подій.

Одним із важливих функціональних етапів роботи програмного додатку є обробка подій, які можуть свідчити про відхилення від нормального стану локального мережевого середовища. Після завершення аналізу виявлених пристроїв система повинна не лише класифікувати мережеві вузли, але й визначати події, що потребують додаткової уваги адміністратора.

Основною метою цього етапу є своєчасне виявлення потенційно небезпечних ситуацій, оцінювання рівня ризику та формування відповідних інформаційних повідомлень для подальшого реагування.

До аномальних ситуацій можуть належати поява нових невідомих пристроїв, зміна відповідності між *IP*- та *MAC*-ідентифікаторами, конфлікти мережевих параметрів, повторювані підозрілі події або інші відхилення від очікуваної структури локального мережевого середовища.

На початковому етапі алгоритму (рис. 3.4) система отримує результати аналізу мережевих вузлів і перевіряє наявність ознак аномальної активності. Якщо відхилень не виявлено, подальша робота продовжується у штатному режимі без формування додаткових попереджень.

У випадку виявлення підозрілої події система виконує оцінювання потенційного рівня ризику. Рівень небезпеки визначається залежно від характеру виявленої події, повторюваності аномалій та потенційного впливу на стабільність і безпеку локального мережевого середовища [6].

Для зручності аналізу доцільно використовувати умовний поділ рівнів загрози:

low — незначне або поодиноке відхилення;

medium — потенційно підозріла подія, що потребує перевірки;

high — подія з ознаками потенційно небезпечного втручання або мережевої атаки.

Після визначення рівня загрози система формує інформаційне повідомлення, яке містить короткий опис виявленої події, її ознаки та службові відомості, необхідні для подальшого аналізу адміністратором.

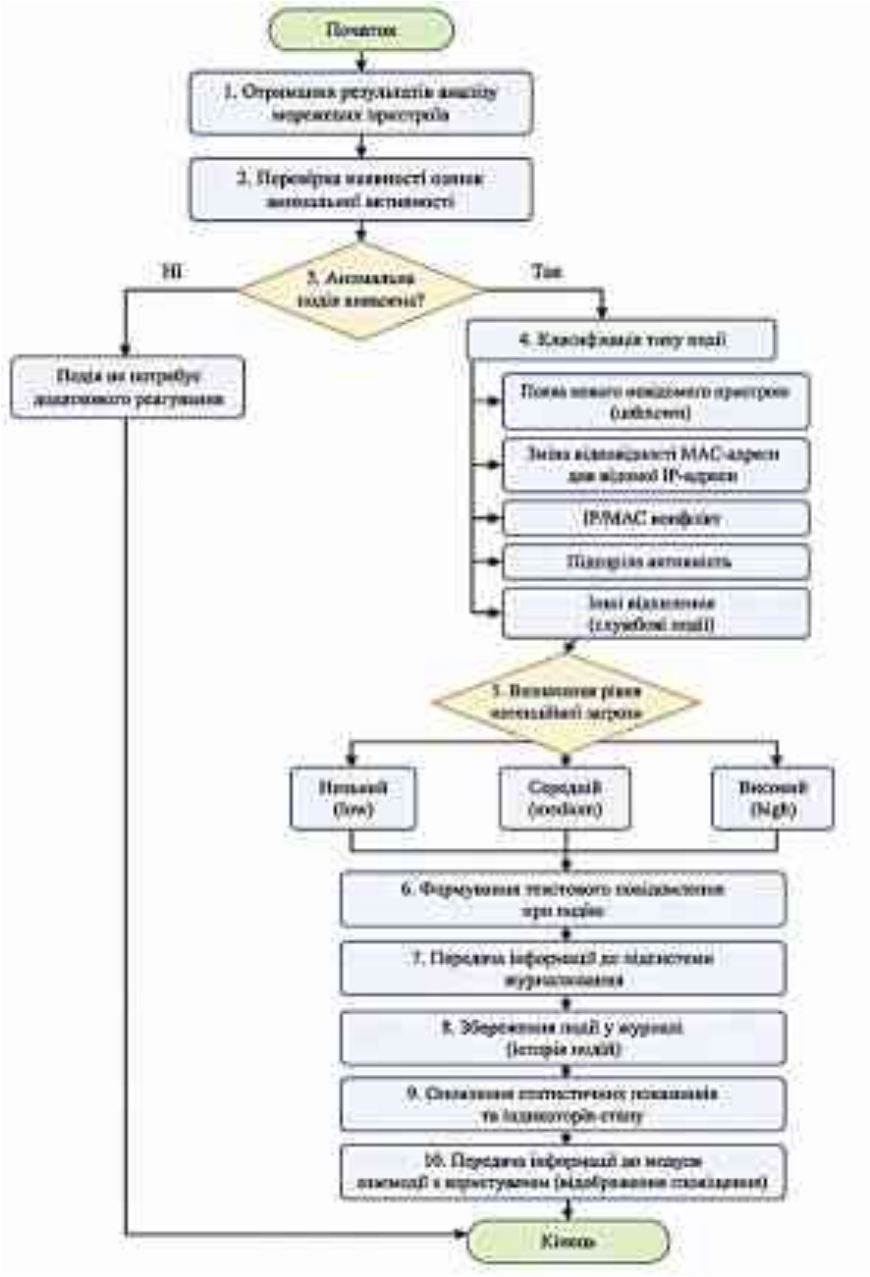

Рис. 3.4 – Алгоритм обробки аномалій та формування сповіщень

Сформовані повідомлення передаються до підсистеми журналювання, де накопичуються для подальшого аналізу стану мережевого середовища та використання під час реагування на потенційні інциденти безпеки.

Запропонований алгоритм дозволяє своєчасно виявляти відхилення у роботі локального мережевого середовища, оцінювати потенційний рівень загрози та забезпечувати інформаційну підтримку адміністратора під час реагування на інциденти інформаційної безпеки.

Для забезпечення ефективного контролю стану локального мережевого середовища важливим є не лише виявлення потенційно підозрілих подій, але й накопичення інформації про результати спостереження, зміни мережевих параметрів та історію раніше зафіксованих інцидентів. З цією метою у структурі програмного додатку передбачено підсистему журналювання подій та формування статистичних показників.

Основним призначенням цього етапу є систематизоване накопичення службової інформації, необхідної для подальшого аналізу стану мережевого середовища, підтримки прийняття адміністративних рішень та оцінювання динаміки потенційних інцидентів інформаційної безпеки [6].

На початковому етапі підсистема отримує інформацію про результати аналізу мережевих пристроїв, класифіковані події та оцінені рівні потенційної загрози. Залежно від характеру отриманих даних система визначає необхідність їх збереження у журналі спостережень.

До журналу подій доцільно вносити відомості про появу нових мережевих вузлів, зміну мережевих параметрів, виявлені аномальні ситуації, службові події системи та інші події, що мають значення для моніторингу локального мережевого середовища.

Окремо повинна накопичуватися історія змін мережевих параметрів, що дозволяє аналізувати нетипові зміни поведінки мережевих вузлів, відстежувати повторювані аномалії та використовувати ці дані під час аналізу потенційних інцидентів.

Крім журналювання подій, підсистема виконує формування статистичних показників стану локального мережевого середовища. До таких показників можуть належати кількість активних пристроїв, кількість нових або невідомих вузлів, кількість подій підвищеного ризику, а також узагальнений індикатор поточного стану мережі.

Формування статистичних показників дозволяє забезпечити швидке оцінювання ситуації в мережі без потреби детального аналізу кожної окремої події, що є важливим для оперативного реагування адміністратора [27].

Алгоритм журналювання подій та формування статистичних показників наведено на рис. 3.5.

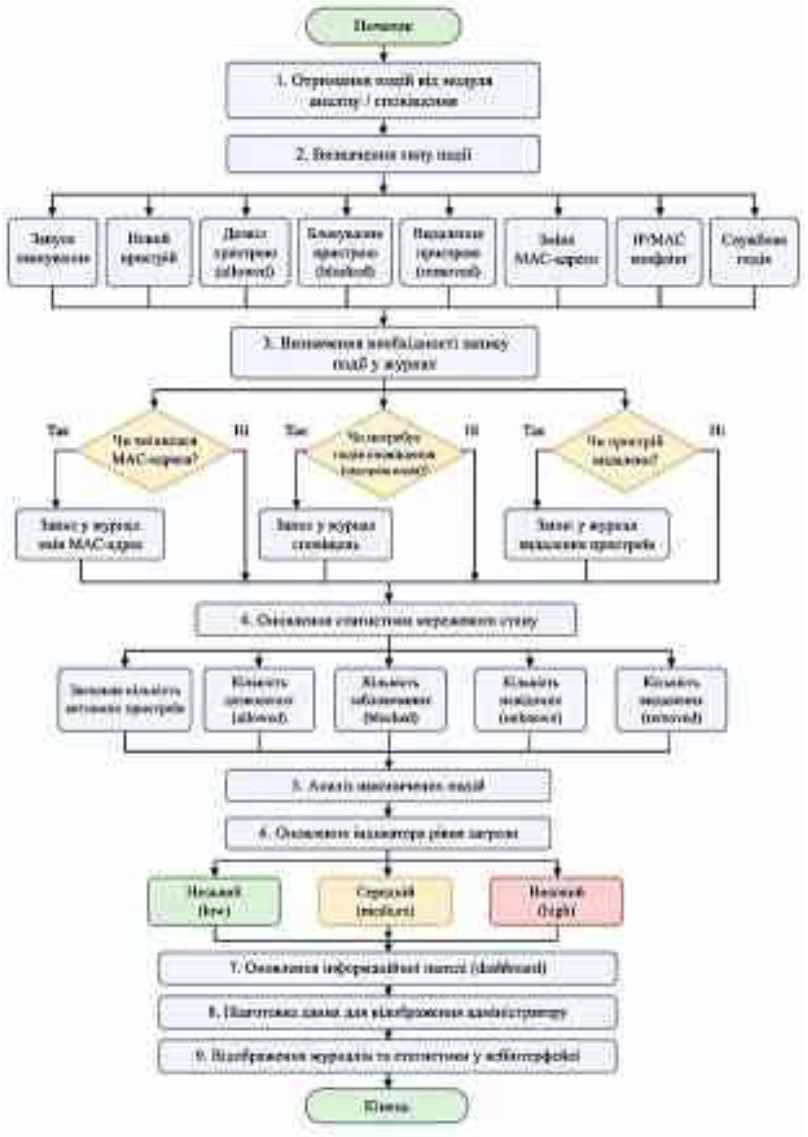

Рис. 3.5 – Алгоритм журналювання подій та формування статистичних показників

Запропонований алгоритм забезпечує накопичення історичних даних про стан локального мережевого середовища, підтримує аналіз змін мережевих параметрів та створює інформаційну основу для оперативного контролю потенційних інцидентів інформаційної безпеки.

3.2. Обгрунтування вибору мови програмування

Одним із важливих етапів розроблення програмного додатку контролю стану локальної мережі та виявлення несанкціонованого доступу є вибір технологічної платформи, яка забезпечить реалізацію необхідного функціоналу, достатню продуктивність, простоту супроводу та можливість подальшого розширення системи [43]. Оскільки розроблюваний програмний додаток поєднує функції мережевого сканування, аналізу мережевих параметрів, журналювання подій, роботи з локальною базою даних і користувацького вебінтерфейсу, технологічний стек мав забезпечувати універсальність, доступність і простоту інтеграції окремих компонентів.

Для реалізації програмного додатку було обрано мову програмування *Python* [39], що зумовлено її практичною придатністю для створення мережевих прикладних систем, високою швидкістю розроблення, зручністю супроводу програмного коду та широкими можливостями інтеграції із сучасними бібліотеками мережевого аналізу й веброзробки. Важливою перевагою обраного підходу є кросплатформеність, що дозволяє використовувати програмний додаток у середовищах *Windows, Linux* або *macOS* за умови встановлення необхідних програмних залежностей.

Розроблений програмний додаток реалізовано як веборієнтований програмний засіб, що поєднує серверну логіку обробки мережевих даних із користувацьким вебінтерфейсом адміністратора. Вибір саме такої архітектури зумовлений потребою забезпечити універсальний доступ до системи без необхідності встановлення окремого клієнтського програмного забезпечення на кожному пристрої. Використання вебінтерфейсу дозволяє адміністратору

працювати із системою через стандартний браузер не лише на персональному комп’ютері, але й на мобільних пристроях, зокрема смартфонах або планшетах, за умови доступу до локальної мережі. Це підвищує зручність практичного використання програмного рішення та розширює можливості оперативного контролю стану мережевого середовища.

Для локального збереження службової інформації використовується система *SQLite* [41], яка є вбудованою файловою системою керування базами даних. Її використання не потребує окремого серверного розгортання чи складного адміністрування, що відповідає концепції створення легкого автономного програмного рішення. Збереження даних у вигляді окремого файлу бази спрощує резервне копіювання, перенесення програмного додатку між пристроями та архівування інформації на зовнішніх носіях. У базі даних зберігаються відомості про мережеві пристрої, журнали подій, історія змін *MAC*-адрес та статистичні показники стану мережевого середовища.

Для реалізації користувацького інтерфейсу та серверної логіки було використано вебфреймворк *Flask*. Його вибір пояснюється відносною простотою інтеграції, компактністю та відсутністю надлишкових архітектурних вимог. Використання *Flask* дозволило реалізувати легкий веборієнтований інтерфейс адміністратора, доступний через стандартний браузер без необхідності створення окремого клієнтського програмного забезпечення [40].

Функція активного виявлення пристроїв у локальному сегменті мережі реалізована за допомогою бібліотеки *Scapy*, яка забезпечує формування та аналіз *ARP*-запитів, отримання відповідей від активних вузлів і базову обробку мережевих пакетів. Це дозволяє програмному додатку виконувати фактичне сканування мережі та отримувати актуальні відомості про *IP*- і *MAC*-адреси виявлених пристроїв [42].

Для обробки мережевого адресного простору використовується модуль *ipaddress*, який забезпечує роботу з адресами у форматі *CIDR*. Виконання допоміжних фонових процесів реалізується за допомогою модуля *threading*,

що дозволяє виконувати сканування без блокування користувацького інтерфейсу. Для взаємодії із системним середовищем використовується *subprocess*, а формування динамічних вебсторінок реалізовано через шаблонізатор *Jinja2* [39].

Візуальна частина інтерфейсу реалізована із застосуванням стандартних вебтехнологій *HTML* та *CSS*, що забезпечує простоту доступу до системи через сучасні браузери. Для коректного функціонування активного мережевого сканування у середовищі *Windows* додатково використовуються драйвери *Npcap* або *WinPcap*, які забезпечують доступ до пакетного рівня мережевого інтерфейсу [44].

У процесі розроблення програмного додатку було підготовлено супровідну експлуатаційну документацію, необхідну для його практичного використання. Зокрема, для спрощення встановлення, налаштування та роботи із системою розроблено Інструкцію користувача, яку наведено у додатку А.

Повний програмний код розробленого додатку, включаючи основні функціональні модулі системи, наведено у додатках Б–Г.

Порівняно з мовами *C++*, *Java* або *C#*, використання *Python* для реалізації навчального прототипу є більш доцільним завдяки меншій складності розроблення, швидшому створенню функціонального програмного рішення та широкій доступності бібліотек для мережевого аналізу й веброзробки. [28].

Таким чином, обраний технологічний стек відповідає поставленим функціональним вимогам та забезпечує реалізацію легкого, автономного й практично придатного програмного рішення для контролю стану локальної мережі та виявлення потенційно несанкціонованого доступу.

3.3. Експериментальне підтвердження працездатності додатку та порядок його використання

Для підтвердження працездатності розробленого програмного додатку було проведено експериментальне дослідження методом натурного моделювання у реальному локальному мережевому середовищі. Такий підхід дозволяє оцінити функціонування програмного засобу в умовах, максимально наближених до практичної експлуатації, із використанням реального мережевого обладнання та фізично підключених пристроїв.

Метою експерименту була перевірка здатності програмного додатку виконувати виявлення активних мережевих вузлів, визначати нові або потенційно несанкціоновані підключення, забезпечувати зміну адміністративних статусів пристроїв та підтримувати контроль за станом локального мережевого сегмента.

Для проведення експерименту було сформовано тестове локальне мережеве середовище, структура якого наведена на рис. 3.6.

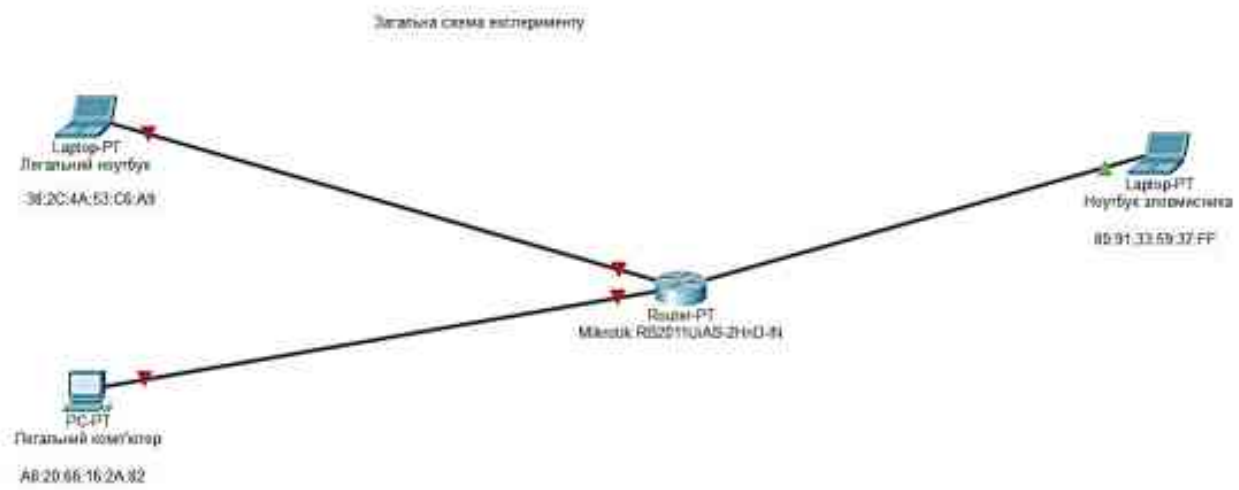

Рис. 3.6 – Загальна схема експериментального середовища для тестування програмного додатку

Експериментальне середовище включало маршрутизатор *MikroTik RB2011UiAS-2HnD-IN*, який виконував функції центрального мережевого вузла, а також три фізичні пристрої, підключені до локальної мережі: легальний ноутбук адміністратора із встановленим програмним додатком, легальний стаціонарний комп'ютер та додатковий ноутбук, який використовувався для моделювання потенційно несанкціонованого підключення.

Характеристики експериментального обладнання наведено у таблиці 3.1.

Таблиця 3.1 – Характеристики обладнання експериментального середовища

	Легальний ноутбук	**Легальний комп'ютер**	**Ноутбук зловмисника**	**Роутер**
Ім'я пристрою	*DESKTOP-FR6P8NG*	*DESKTOP-F55D041*	*DESKTOP-CL9SMS8*	*Router.lan*
Процесор	*Intel i*7 4510*U* 2.00*GHz*	*Intel i*7 3615*QM* 2.30*GHz*	*Intel Pentium* 4417*U* 2.30*GHz*	*Atheros* 600 *Mz*
Оперативна память	8 Гб	16 Гб	4Гб	128 Мб
Операційна система	*Windows* 11	*Windows* 10	*Windows* 11	*Mikrotik RouterOS*

На першому етапі експерименту до локальної мережі було підключено лише легітимні пристрої — ноутбук адміністратора та стаціонарний комп'ютер. Після запуску програмного додатку виконувалося реальне *ARP*-сканування локального сегмента мережі з метою виявлення активних вузлів.

Результат первинного сканування наведено на рис. 3.7.

За результатами сканування програмний додаток успішно виявив активні мережеві вузли, визначив локальний пристрій адміністратора та зафіксував нові мережеві об'єкти, що потребували подальшої класифікації.

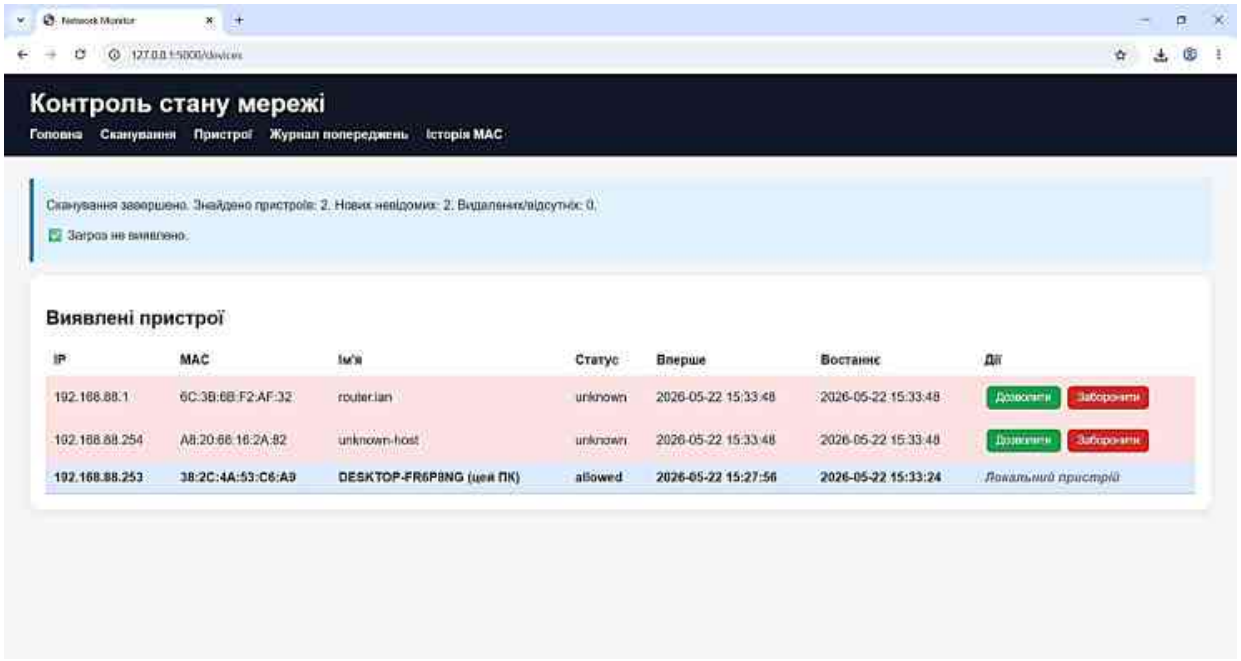

Рис. 3.7 Результат сканування локальної мережі з виявленням активних пристроїв

На наступному етапі адміністратор виконав підтвердження легітимності виявленого стаціонарного комп’ютера шляхом зміни його статусу на *allowed*. Результат цієї операції наведено на рис. 3.8.

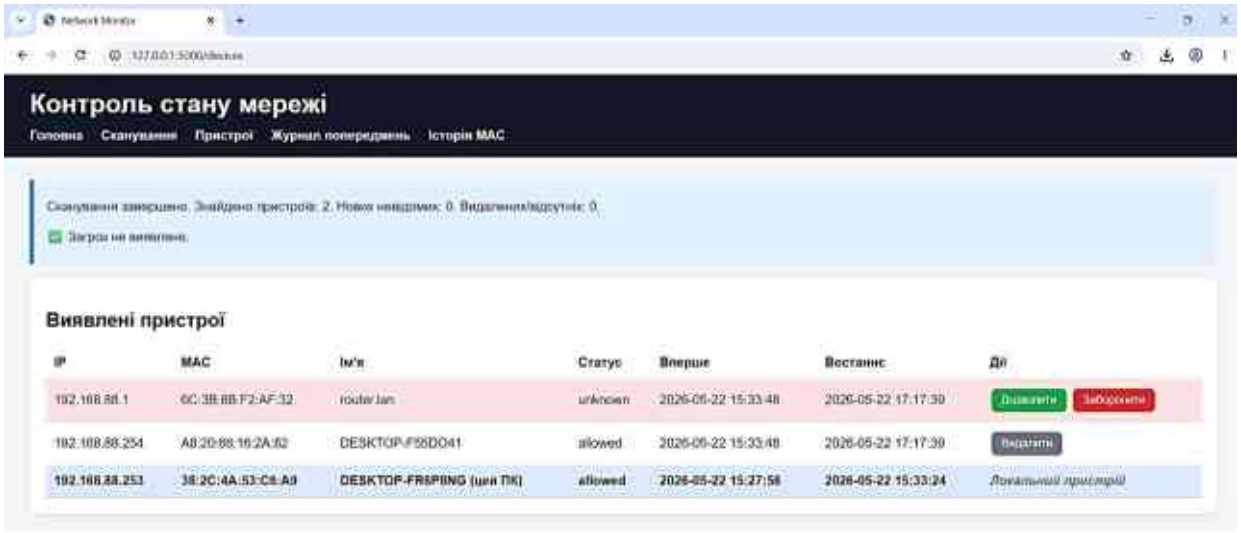

Рис. 3.8 – Підтвердження легітимного мережевого пристрою

Після цього аналогічну процедуру було виконано для мережевого маршрутизатора, який також був ідентифікований як легітимний елемент локальної інфраструктури. Відповідний результат наведено на рис. 3.9.

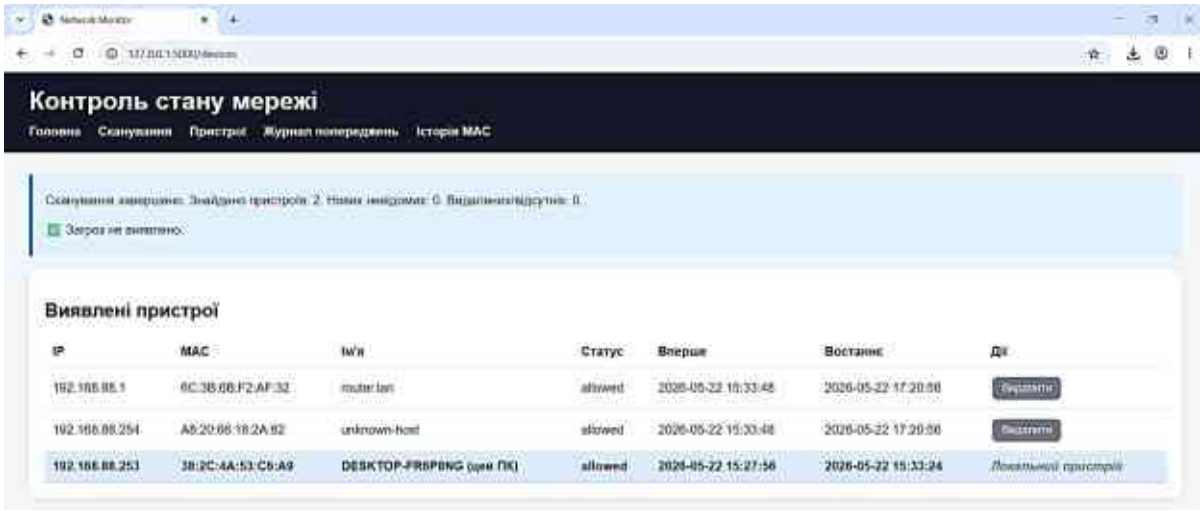

Рис. 3.9 – Додавання легітимного мережевого маршрутизатора до переліку довірених пристроїв

На завершальному етапі експерименту до локального сегмента було підключено додатковий ноутбук, який моделював несанкціонований пристрій. Після повторного запуску мережевого сканування система зафіксувала появу нового вузла, що раніше був відсутній у базі даних, та автоматично класифікувала його як невідомий пристрій зі статусом *unknown*.

Результат виявлення нового підключення наведено на рис. 3.10.

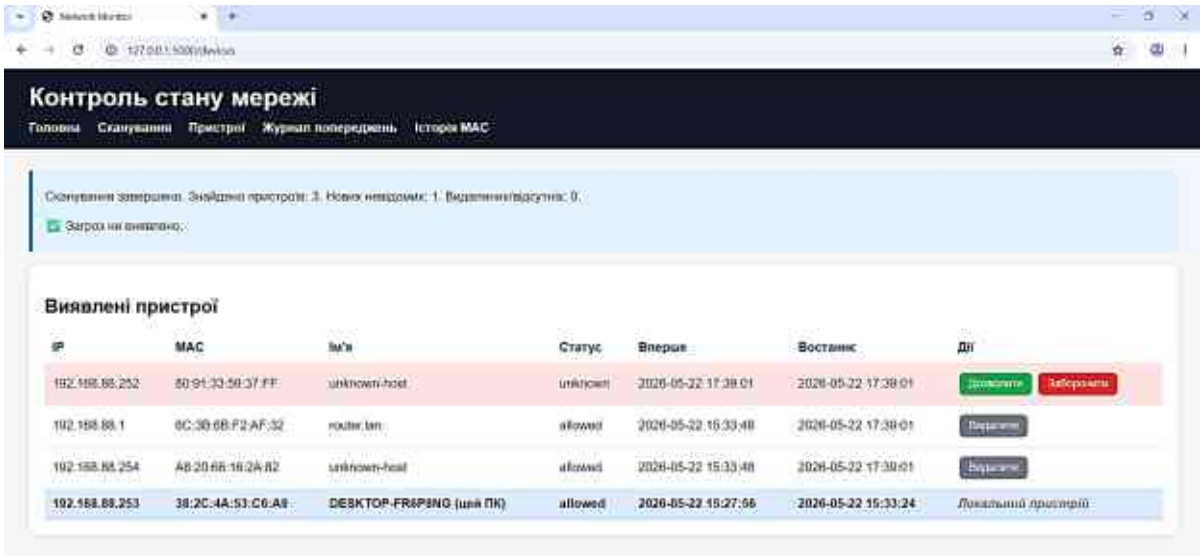

Рис. 3.10 – Виявлення нового потенційно несанкціонованого пристрою

Після аналізу результатів адміністратор виконав блокування підозрілого вузла шляхом переведення його у статус *blocked*. Після цього програмний додаток коректно оновив стан пристрою у базі даних та відобразив змінений статус у вебінтерфейсі.

Результат блокування наведено на рис. 3.11.

Результати натурного експерименту підтвердили працездатність розробленого програмного додатку в умовах реального локального мережевого середовища. Система продемонструвала здатність коректно виявляти активні мережеві вузли, ідентифікувати нові підключення, підтримувати механізм керування статусами пристроїв та забезпечувати базовий контроль потенційно несанкціонованого доступу до локальної мережі.

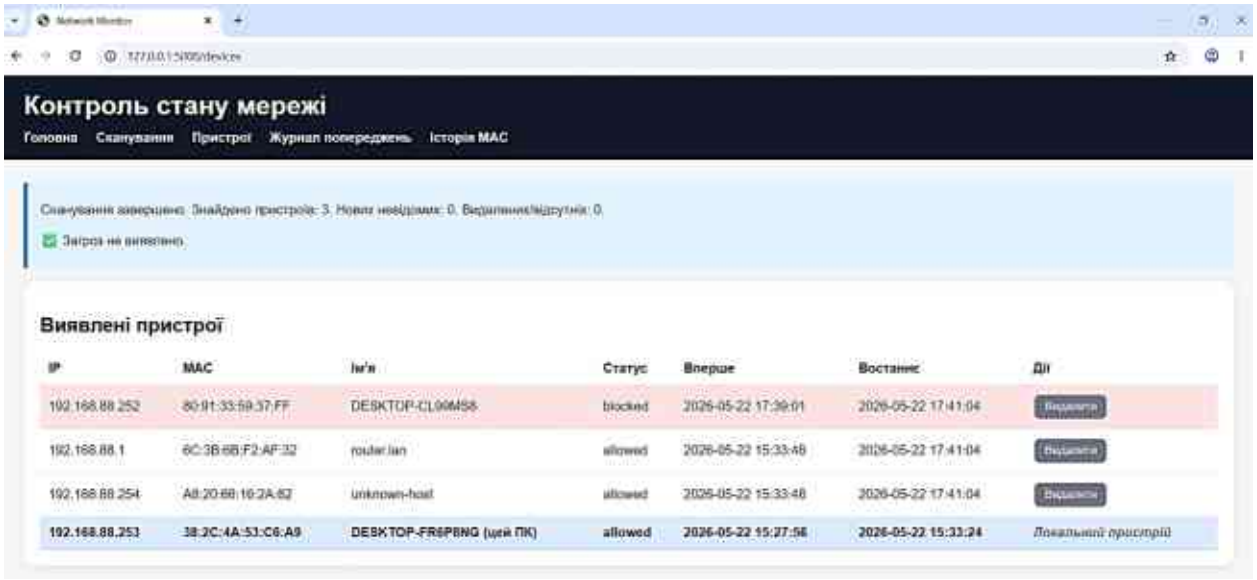

Рис. 3.11 – Блокування потенційно несанкціонованого пристрою

Перед практичним застосуванням розробленого програмного додатку необхідно виконати попередню підготовку програмного середовища та забезпечити доступ до локального мережевого сегмента, у межах якого здійснюватиметься моніторинг.

Для коректної роботи системи треба виконати встановлення необхідних програмних компонентів, налаштування середовища виконання та підготовку мережевого інтерфейсу для роботи в режимі реального сканування. Після

цього програмний додаток може бути запущений у локальному середовищі з подальшим доступом до його функціональних можливостей через користувацький вебінтерфейс.

Детальний порядок встановлення, налаштування та запуску програмного додатку наведено в додатку А (Інструкція користувача).

Після підготовки програмного середовища та запуску системи адміністратор отримує доступ до вебінтерфейсу програмного додатку, через який здійснюється контроль стану локального мережевого середовища, аналіз виявлених пристроїв та реагування на потенційні інциденти безпеки.

Початковим етапом роботи є доступ до головної сторінки системи, де відображаються узагальнені статистичні показники стану мережі, кількість виявлених пристроїв та поточний індикатор рівня потенційної загрози (рис. 3.12).

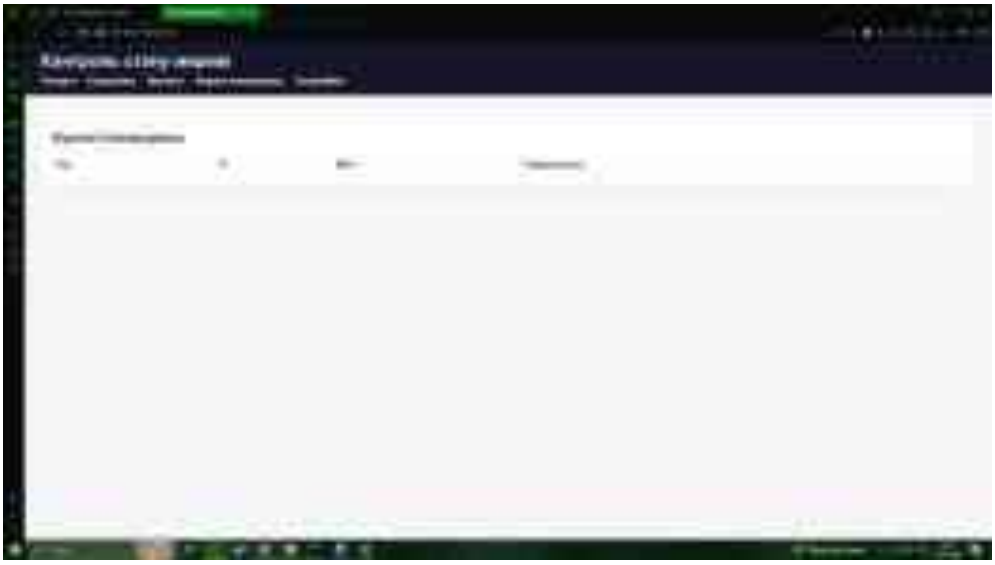

Рис. 3.12 – Головна сторінка програмного додатку після запуску

Для запуску процедури мережевого аналізу користувач переходить до сторінки сканування, де задаються параметри перевірки локального сегмента мережі та запускається процедура виявлення активних вузлів (рис.3.13).

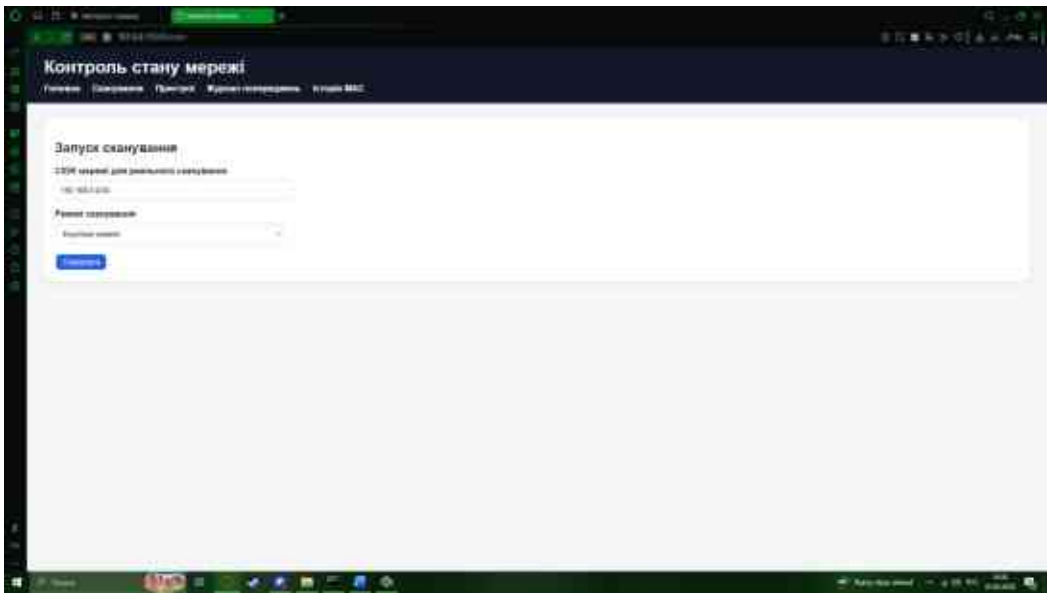

Рис. 3.13 – Запуск процедури мережевого сканування

Після завершення аналізу адміністратор переходить до сторінки перегляду пристроїв, де відображається перелік виявлених мережевих вузлів із відповідними *IP*- та *MAC*-адресами, а також їхніми адміністративними статусами (рис. 3.14).

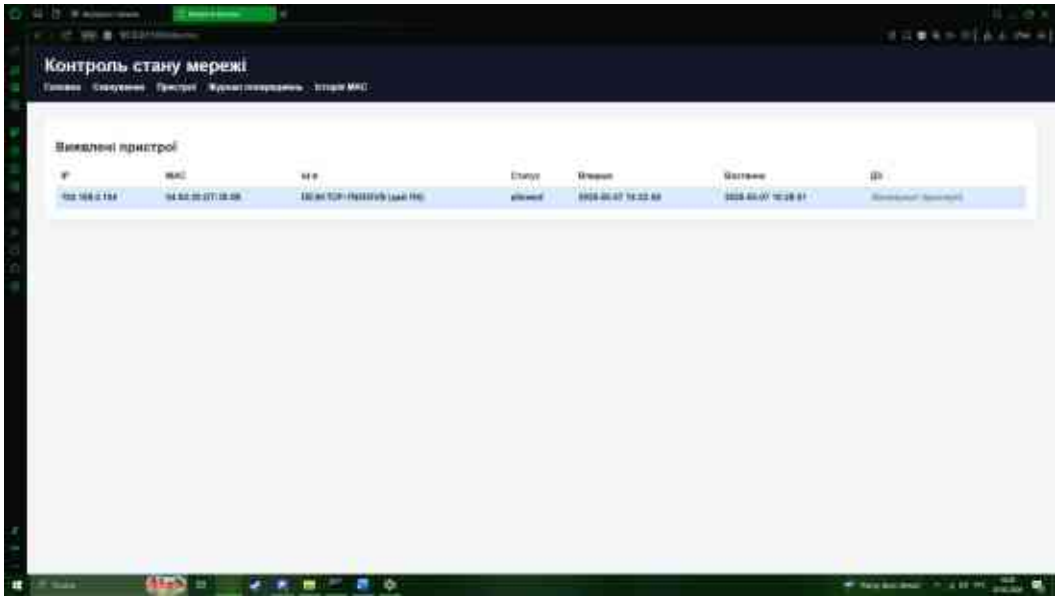

Рис. 3.14 – Перелік виявлених мережевих пристроїв

За результатами аналізу користувач може підтвердити легітимність пристрою, перевести його до переліку довірених або, у разі виявлення

підозрілої активності, обмежити його використання шляхом зміни адміністративного статусу.

Для аналізу зафіксованих інцидентів передбачено журнал подій, у якому відображаються системні попередження, повідомлення про нові пристрої та інші події інформаційної безпеки (рис.3.15).

Рис. 3.15 – Журнал подій інформаційної безпеки

Додатково система надає можливість аналізу історії змін *MAC*-ідентифікаторів, що може бути використано для виявлення нетипових змін мережевих параметрів або аналізу потенційних мережевих аномалій (рис. 3.16).

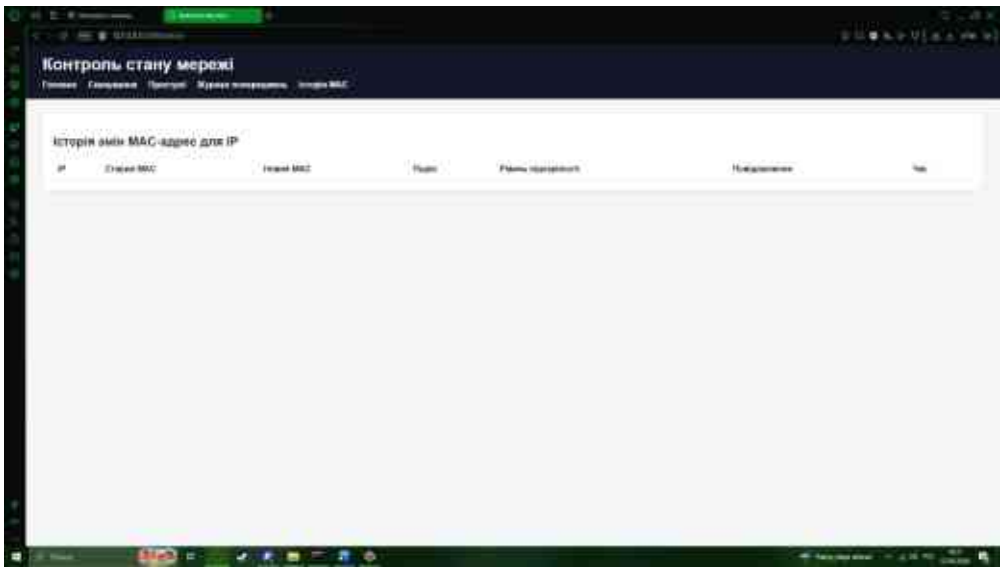

Рис. 3.16 – Історія змін *MAC*-адрес

Під час практичного використання системи моніторингу локальної мережі важливим є не лише виявлення потенційно небезпечних подій, але й своєчасне прийняття рішень щодо реагування на інциденти інформаційної безпеки. Розроблений програмний додаток формує інформаційну основу для такого реагування шляхом виявлення підозрілих подій, аналізу стану мережевого середовища та накопичення журналів спостережень.

Реагування на виявлені інциденти повинно здійснюватися залежно від характеру потенційної загрози.

У випадку появи невідомого мережевого вузла доцільним є встановлення його походження та перевірка відповідності фактичній конфігурації локального сегмента. Якщо походження пристрою не підтверджено, необхідно розглядати його як потенційно небажаний елемент мережевого середовища.

У разі фіксації аномальних змін мережевих параметрів, зокрема невідповідності між *IP*- та *MAC*-ідентифікаторами, необхідно провести додаткову перевірку мережевої конфігурації, оскільки такі ситуації можуть бути пов’язані як із технічними змінами обладнання, так і з ознаками мережевих атак.

Якщо система фіксує повторювані підозрілі події або підвищення рівня загрози, доцільно виконати додатковий аудит локального сегмента мережі, повторну перевірку активних вузлів та аналіз поточного стану мережевого обладнання.

У випадках, коли виявлені ознаки можуть свідчити про активну атаку або несанкціоноване втручання, рекомендованим є застосування додаткових заходів захисту, зокрема ізоляції підозрілого вузла, обмеження доступу до локального сегмента або проведення поглибленого аналізу мережевого трафіку.

Рекомендовані дії адміністратора наведено у таблиці 3.2.

Таблиця 3.2 – Рекомендовані дії адміністратора при виявленні інцидентів

Тип події	Ознака	Рекомендовані дії
Новий пристрій	статус *unknown*	Перевірити походження пристрою; дозволити або заблокувати
Зміна *MAC*-адреси	невідповідність *IP/MAC*	Перевірити журнал історії змін; оцінити можливість *ARP- spoofing*
IP/MAC-конфлікт	дублювання мережевих параметрів	Виконати повторне сканування; перевірити мережеву конфігурацію
Високий рівень ризику	*high*	Провести додаткову перевірку; обмежити доступ підозрілого вузла
Повторні підозрілі події	множинні попередження	Виконати аудит локального сегмента мережі

Таким чином, розроблений програмний додаток забезпечує не лише виявлення потенційно небезпечних подій, але й створює інформаційну основу для оперативного реагування адміністратора на інциденти інформаційної безпеки.

Висновок до третього розділу

У третьому розділі кваліфікаційної роботи виконано проєктування, програмну реалізацію та експериментальне підтвердження працездатності програмного додатку контролю стану локальної мережі та виявлення несанкціонованого доступу.

У межах розділу розроблено функціональну структуру програмного додатку, алгоритми збору мережевих даних, аналізу виявлених пристроїв, обробки аномальних подій та журналювання результатів моніторингу. Обґрунтовано вибір технологічного стеку реалізації, що включає мову *Python*, вебфреймворк *Flask*, бібліотеку *Scapy* та систему збереження даних *SQLite*.

Проведене експериментальне дослідження методом натурного моделювання у реальному локальному мережевому середовищі підтвердило працездатність розробленого програмного додатку. Результати тестування

показали здатність системи виявляти активні мережеві пристрої, фіксувати появу нових підключень, підтримувати контроль за статусами мережевих вузлів та забезпечувати інформаційну підтримку реагування на потенційні інциденти безпеки.

Таким чином, поставлені завдання третього розділу виконано повністю, а розроблений програмний додаток підтвердив практичну придатність як засіб контролю стану локальної мережі та виявлення потенційно несанкціонованого доступу.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне завдання розроблення програмного додатку контролю стану локальної мережі та виявлення потенційно несанкціонованого доступу до неї.

У процесі виконання роботи проведено аналіз сучасних мережевих кіберзагроз, пов’язаних із несанкціонованим доступом до локальних мереж, а також досліджено існуючі програмні засоби мережевого моніторингу. За результатами аналізу встановлено, що наявні програмні рішення часто характеризуються складністю впровадження, надлишковою функціональністю або значними ресурсними вимогами, що обмежує їх доцільність для невеликих локальних мереж.

У роботі розроблено функціональну структуру програмного додатку, яка включає модулі збору мережевих даних, аналізу та детектування пристроїв, обробки аномалій, журналювання подій та взаємодії з користувачем. Запропоновано алгоритми контролю мережевого середовища, що забезпечують виявлення активних вузлів, аналіз мережевих параметрів, оцінювання потенційно небезпечних подій і формування інформаційної підтримки адміністратора.

Обґрунтовано вибір засобів програмної реалізації та створено програмний додаток, що забезпечує контроль локального мережевого середовища, виявлення нових або підозрілих мережевих пристроїв, аналіз змін *IP*- та *MAC*-ідентифікаторів, ведення журналів подій і формування статистичних показників.

Експериментальне підтвердження працездатності програмного додатку проведено методом натурного моделювання у реальному локальному мережевому середовищі із використанням фізично підключених мережевих пристроїв. Результати тестування підтвердили здатність системи коректно виявляти активні мережеві вузли, фіксувати появу нових підключень,

підтримувати контроль статусів пристроїв та забезпечувати інформаційну підтримку реагування на потенційні інциденти інформаційної безпеки.

Отже, поставлена мета роботи досягнута, а визначені завдання виконані у повному обсязі. Розроблений програмний додаток є практично придатним для використання як засіб контролю стану локального мережевого середовища та виявлення потенційно несанкціонованого доступу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойко А.О., Марущак А.І. Безпека комп'ютерних мереж: навч. посіб. – Київ: Видавництво «Центр учбової літератури», 2020. — 256 с.

2. https://techexpert.ua/solarwinds-attack-security/

3. https://ussd.org.ua/2021/06/03/dokladno-pro-kiberataku-na-colonial-pipeline/

4. https://eurepoc.eu/wp-content/uploads/2023/10/KA-SAT-Viasat-MaCI.pdf

5. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги. – Київ: ДП «УкрНДНЦ».

6. Scarfone K., Mell P. Guide to Intrusion Detection and Prevention Systems (IDPS)». NIST Special Publication 800-94. – Gaithersbug: National Institute of Standards and Technology, 2022. – 127 p.

7. NIST SP 800-61, редакція 3. Рекомендації щодо реагування на інциденти та міркування щодо управління ризиками кібербезпеки: профіль спільноти CSF 2.0 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r3.pdf

8. Палій В.В. Мережеві технології та безпека: практикум. – Львів: Видавництво Львівської політехніки. 2021. – 180 с.

9. Zabbix Documentation [Електронний ресурс]. — https://www.zabbix.com/documentation/current/en/manual

10. Nagios Core Documentation [Електронний ресурс]. — https://www.nagios.org/documentation

11. PRTG Network Monitor Documentation [Електронний ресурс]. — https://www.paessler.com/manuals/prtg

12. Security Onion Documentation [Електронний ресурс]. — https://docs.securityonion.net/

13. Suricata IDS Documentation [Електронний ресурс]. — https://docs.suricata.io

14. Zeek Network Security Monitor Documentation [Електронний ресурс]. — https://docs.zeek.org

15. Analysis of the Cyber Attack on the Ukrainian Power Grid [Електронний ресурс]. — Режим доступу: https://www.isa.org/intech-home/2017/march-april/features/ukrainian-power-grids-cyberattack

16. Kyiv cyberattack 2016 [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/2016_Kyiv_cyberattack

17. Plummer D. Ethernet Address Resolution Protocol (ARP). RFC 826. https://datatracker.ietf.org/doc/html/rfc826

18. Droms R. Dynamic Host Configuration Protocol. RFC 2131. https://datatracker.ietf.org/doc/html/rfc2131

19. Wireshark Foundation. Wireshark User Guide [Електронний ресурс]. — Режим доступу: https://www.wireshark.org/docs/wsug_html_chunked/

20. Postel J. Internet Control Message Protocol. RFC 792. https://datatracker.ietf.org/doc/html/rfc792

21. arp-scan documentation https://github.com/royhills/arp-scan

22. Lyon G. Nmap Network Scanning. — Insecure.org. https://nmap.org/book/man.html

23. Kent K., Souppaya M. Guide to Computer Security Log Management. NIST SP 800-92 [Електронний ресурс]. — Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-92/final

24. Sommer R., Paxson V. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection // IEEE Symposium on Security and Privacy.

25. Behl B., Behl K. Cybersecurity and Cyberwar: What Everyone Needs to Know. — Oxford University Press.

26. ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) [Електронний ресурс]. — Режим доступу: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

27. Stallings W. Network Security Essentials: Applications and Standards. — Pearson Education. — 6th ed. — Pearson, 2020. — 672 p.

28. OWASP Foundation. OWASP Top Ten Web Application Security Risks [Електронний ресурс]. — Режим доступу: https://owasp.org/www-project-top-ten/

29. ISO/IEC 27001:2015 Information security management systems — Requirements

30. NIST Digital Identity Guidelines SP 800-63B [Електронний ресурс]. — Режим доступу: https://pages.nist.gov/800-63-3/sp800-63b.html

31. ENISA Threat Landscape 2024 [Електронний ресурс]. — Режим доступу: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

32. https://www.academia.edu/31504245/A_Taxonomy_of_DDoS_Attack_and_DDoS_Defense_Mechanisms

33. ENISA Malware Threat Landscape Report [Електронний ресурс]. — Режим доступу: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025

34. MITRE ATT&CK Framework Enterprise Matrix [Електронний ресурс]. — Режим доступу: https://attack.mitre.org/matrices/enterprise/

35. MITRE ATT&CK Data Exfiltration Techniques [Електронний ресурс]. — Режим доступу:https://attack.mitre.org/tactics/TA0010/

36. https://www.vendr.com/marketplace/nagios

37. https://www.paessler.com/pricing

38. https://www.solarwinds.com/pricing

39. Python Software Foundation. Python 3 Documentation [Електронний ресурс]. — Режим доступу: https://docs.python.org/3/

40. Flask Documentation (Pallets Projects) [Електронний ресурс]. — Режим доступу: https://flask.palletsprojects.com/

41. SQLite Documentation [Електронний ресурс]. — Режим доступу: https://www.sqlite.org/docs.html

42. Scapy Documentation [Електронний ресурс]. — Режим доступу: https://scapy.readthedocs.io/

43. Sommerville I. Software Engineering. — 10th ed. — Pearson, 2016. — 816 p.

44. Npcap Documentation [Електронний ресурс]. — Режим доступу: https://npcap.com/

45. Northcutt S., Novak J. Network Intrusion Detection. Sams Publishing , 2002. - 490 p.

Додаток А

ІНСТРУКЦІЯ КОРИСТУВАЧА

Додаток контролю стану локальної мережі та виявлення підозрілих IP/MAC-подій

Зміст

1. Загальні відомості

Це навчальний веб-додаток для контролю стану локальної мережі. Програма дозволяє виконувати сканування мережі, формувати перелік виявлених пристроїв, призначати їм статуси, вести журнал подій та виявляти підозрілі зміни відповідності IP-адрес і MAC-адрес.

Додаток розроблено як прототип для бакалаврської роботи. Він має два основні режими роботи: реальне ARP-сканування локальної мережі та емуляцію мережі без потреби у фізичному доступі до мережевого обладнання.

Можливість	**Опис**
Облік пристроїв	Формування списку виявлених пристроїв із IP, MAC, іменем, статусом і часом появи.
Керування статусами	Адміністратор може дозволити, заборонити або видалити пристрій.
Журнал подій	Фіксує старт сканування, виявлення пристроїв, дозвіл, заборону, видалення та помилки.
Індикатор загроз	Показує безпечний, підозрілий або небезпечний стан на основі останніх подій.
Історія MAC	Фіксує випадки, коли одна IP-адреса пов’язується з різними MAC-адресами.
Stateful-емуляція	Створює стабільну тестову мережу в SQLite, де більшість пристроїв залишаються між сканами.

2. Системні вимоги

Компонент	**Вимога**
Python	Версія 3.10 або новіша.
Операційна система	Windows 10/11, Linux або macOS.
Браузер	Будь-який сучасний браузер: Chrome, Edge, Firefox тощо.
Доступ до мережі	Не обов’язковий для режиму емуляції. Потрібний для реального сканування.
Додатковий драйвер для Windows	Npcap або WinPcap для реального ARP-сканування.

3. Склад проєкту

Файл або папка	**Призначення**
app.py	Головний Flask-додаток, маршрути сторінок і обробка дій користувача.
database.py	Робота з SQLite, пристроями, журналом, історією MAC та емуляцією мережі.
scanner.py	Функція реального ARP-сканування локальної мережі.
config.py	Базові параметри: шлях до бази даних, CIDR за замовчуванням, режим сканування.
templates/	HTML-шаблони сторінок інтерфейсу.
static/style.css	Стилі веб-інтерфейсу.
requirements.txt	Список Python-залежностей.

4. Встановлення та запуск

Нижче наведено типовий порядок запуску програми у Windows через PowerShell або CMD.

Розпакувати архів з проєктом у зручну папку, наприклад **D:\Projects\network_monitor_project**.

Відкрити термінал у папці проєкту.

Створити віртуальне середовище Python.

python -m venv venv

Активувати віртуальне середовище.

venv\Scripts\activate

Якщо PowerShell блокує виконання скриптів, можна або активувати середовище через CMD, або дозволити локальні скрипти для поточного користувача:

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

Встановити залежності.

pip install -r requirements.txt

Запустити додаток.

python app.py

Відкрити у браузері адресу:

http://127.0.0.1:5000

Запуск без активації середовища

Якщо активація віртуального середовища не використовується, програму можна запускати напряму через Python із папки venv:

venv\Scripts\python app.py

5. Огляд інтерфейсу

Після запуску користувач отримує веб-інтерфейс із верхнім меню. Основні сторінки:

Сторінка	**Призначення**
Головна	Загальна статистика, індикатор загроз, останні повідомлення журналу, очищення бази.
Сканування	Запуск реального або емуляційного сканування мережі.
Пристрої	Перегляд виявлених пристроїв та виконання дій: дозволити, заборонити, видалити.
Журнал попереджень	Перелік усіх зафіксованих подій системи.
Історія MAC	Події зміни MAC-адреси для однієї IP-адреси.

6. Робота з головною сторінкою

Головна сторінка використовується для швидкого контролю стану системи. На ній відображається індикатор загроз і картки статистики.

Елемент	Значення
Індикатор загроз	Показує поточний стан: загроз не виявлено, підозріла активність, ● потенційна загроза.
Усього пристроїв	Кількість пристроїв, що є у поточній таблиці devices.
Дозволені	Кількість пристроїв зі статусом allowed.
Невідомі	Кількість пристроїв зі статусом unknown.
Заборонені	Кількість пристроїв зі статусом blocked.
Видалені	Кількість пристроїв, які були видалені з бази та записані в таблицю deleted_devices.

Індикатор загроз формується на основі останніх повідомлень журналу. Події з позначкою ● мають вищий пріоритет, ніж .

7. Запуск сканування

Сторінка «Сканування» дозволяє вибрати режим роботи та запустити перевірку мережі. Під час виконання сканування кнопка стає неактивною, а на сторінці відображається індикатор очікування.

7.1. Поле CIDR

CIDR — це діапазон адрес, який потрібно перевірити при реальному скануванні. Програма автоматично формує значення на основі IP-адреси локального ПК. Наприклад:

IP локального ПК	CIDR для сканування
192.168.1.45	192.168.1.0/24
10.0.0.23	10.0.0.0/24
127.0.0.1	192.168.0.0/24

7.2. Режими сканування

Режим	Коли використовувати	Особливості
Емуляція мережі	Для тестування без доступу до реальної мережі.	Використовує стабільну віртуальну топологію, що зберігається в SQLite.
Сканування фактичної мережі	Для перевірки реальної локальної мережі.	Потребує мережевого доступу та, на Windows, драйвера Npcap/WinPcap.

7.3. Порядок запуску сканування

Перейти до сторінки «Сканування».

Перевірити або відредагувати CIDR мережі.

Вибрати режим: «Емуляція мережі» або «Сканування фактичної мережі».

Натиснути кнопку «Сканувати».

Дочекатися завершення. Кнопка під час сканування блокується, а під нею відображається спіннер.

Після завершення система перенаправить користувача на сторінку «Пристрої».

8. Робота зі списком пристроїв

Сторінка «Пристрої» містить таблицю з усіма виявленими або зареєстрованими пристроями. Для кожного пристрою показуються IP-адреса, MAC-адреса, ім’я, статус, час першого і останнього виявлення та доступні дії.

Статус	Значення
unknown	Новий або невизначений пристрій, щодо якого адміністратор ще не ухвалив рішення.
allowed	Дозволений пристрій. Після повторних сканів статус зберігається.
blocked	Заборонений пристрій. Після повторних сканів статус зберігається.
removed	Пристрій вважається відсутнім або видаленим. У режимі емуляції статус ставиться після кількох пропущених сканів.

8.1. Дії користувача з пристроями

Поточний стан	Доступні кнопки	Результат
unknown	Дозволити	Статус змінюється на allowed, дія записується в журнал.
unknown	Заборонити	Статус змінюється на blocked, дія записується в журнал.
allowed або blocked	Видалити	Пристрій видаляється з основного списку, записується в deleted_devices і журнал.
локальний ПК	Немає кнопок керування	Локальний пристрій захищено від випадкового видалення через інтерфейс.
removed	Немає активних дій	Відображається повідомлення, що пристрій видалено.

8.2. Локальний пристрій

Локальний ПК автоматично додається до бази при запуску програми та після очищення бази. У списку пристроїв він має позначку «(цей ПК)» і окреме візуальне підсвічування. Це дозволяє відрізняти пристрій адміністратора від інших елементів мережі.

9. Журнал подій

Сторінка «Журнал попереджень» фактично виконує роль журналу подій. Вона фіксує не лише загрози, а й усі важливі дії користувача та системи.

Тип події	Приклад повідомлення
Старт сканування	З локального пристрою запущено нове сканування мережі.
Новий пристрій	Виявлено новий невідомий пристрій.
Дозвіл пристрою	Пристрій дозволено адміністратором.
Заборона пристрою	Пристрій заборонено адміністратором.
Видалення пристрою	Адміністратор видалив пристрій з бази.
Тимчасова відсутність	Пристрій тимчасово не виявлено у скані.
IP/MAC-конфлікт	Для IP-адреси виявлено зміну MAC-адреси.
Помилка	Помилка сканування або службова помилка виконання.

9.1. Індикатори в журналі

Позначка	Рівень	Пояснення
□	Низький	Подія не становить безпосередньої загрози.
□	Середній	Виявлена підозріла активність або нетипова ознака.
●	Високий	Виявлена потенційно небезпечна подія, наприклад конфлікт IP/MAC з кількома факторами ризику.

10. Історія MAC-адрес

Сторінка «Історія MAC» використовується для аналізу ситуацій, коли одна IP-адреса була пов'язана з різними MAC-адресами. Така ситуація може бути нормальною для деяких мережевих сценаріїв, але також може свідчити про ARP spoofing, зміну мережевого адаптера, VPN, віртуалізацію або конфлікт адрес.

Поле	Опис
IP	IP-адреса, для якої зафіксовано зміну MAC.
Старий MAC	MAC-адреса, яка раніше була пов'язана з цією IP.
Новий MAC	MAC-адреса, знайдена під час нового сканування.
Подія	Тип події, наприклад mac_changed_for_ip.
Рівень підозрілості	low, medium або high.
Повідомлення	Пояснення події.
Час	Дата й час фіксації.

10.1. Оцінка підозрілості

Рівень підозрілості визначається евристично. Система враховує, чи є пристрій локальним ПК, чи має MAC ознаки локально згенерованої адреси, чи належить IP до типових адрес шлюзів .1 або .2, а також чи накопичуються кілька факторів одночасно.

Рівень	Типова ситуація
low	Звичайна подія або локальний ПК.
medium	Один фактор ризику: наприклад MAC починається з 02 або IP схожий на адресу шлюзу.
high	Кілька факторів ризику одночасно або подія виглядає як можлива підміна.

11. Реалістична емуляція мережі

Режим емуляції призначений для тестування програми без доступу до реальної мережі. На відміну від випадкової генерації, поточна реалізація зберігає стан мережі в SQLite. Це означає, що між сканами більшість пристроїв залишаються тими самими, а зміни відбуваються поступово.

11.1. Типи емульованих пристроїв

Тип	Приклади	Поведінка
core	Local PC, Router, Backup-Router	Майже завжди присутні у мережі.
mobile	Laptop, Smartphone, Tablet	Можуть тимчасово зникати й повертатися.
guest	Guest-Phone, IoT-Device, Unknown-Device	З'являються періодично та можуть швидко зникати.
suspicious	Spoofed-Device, Fake-Router	Можуть створювати підозрілу IP/MAC-подію.

11.2. Принцип роботи емуляції

При першому емуляційному скануванні створюється базова топологія мережі.

До топології завжди додаються локальний ПК і роутер.

Додатково створюються мобільні, гостьові й службові пристрої.

Кожен наступний скан використовує попередню топологію з таблиці simulated_devices.

Для кожного пристрою застосовується коефіцієнт стабільності: стабільні пристрої частіше присутні у скані.

Нові гостьові пристрої можуть додаватися рідко, щоб імітувати реальну поведінку мережі.

Іноді система створює IP/MAC-конфлікт для демонстрації підозрілої активності.

11.3. Логіка відсутніх пристроїв

Щоб пристрої не ставали removed після одного випадкового пропуску, використовується лічильник missed_scans. Якщо пристрій не знайдено під час поточного сканування, лічильник збільшується. Якщо пристрій знову знайдено — лічильник скидається до нуля.

Кількість пропусків	**Стан**
0	Пристрій активний, знайдений у поточному скані.
1–2	Пристрій тимчасово не виявлено, але ще не видалений.
3 або більше	Пристрій отримує статус removed.

12. Реальне сканування мережі

Реальне сканування виконується за допомогою ARP-запитів. У Windows для цього зазвичай потрібен Npcap або WinPcap. Без відповідного драйвера може з’явитися помилка про недоступність sniffing/sending packets at layer 2.

12.1. Підготовка Windows

Встановити Npcap з офіційного сайту.

Під час встановлення увімкнути параметр WinPcap API-compatible mode.

Перезапустити комп’ютер за потреби.

Запустити термінал від імені адміністратора.

Запустити програму та виконати сканування фактичної мережі.

12.2. Особливості інтерпретації результатів

Якщо одна IP-адреса має різні MAC-адреси, це не завжди означає атаку. Можливі причини: зміна мережевого адаптера, Wi-Fi/Ethernet, віртуалізація, VPN, ARP-кеш або реальна підміна. Саме тому система не видаляє такі дані автоматично, а фіксує подію в історії MAC і журналі.

13. Очищення бази даних

На головній сторінці є кнопка «Очистити базу для нового тесту». Вона використовується для повернення програми у початковий стан.

Що очищається	**Пояснення**
devices	Основний список пристроїв.
alerts	Журнал подій.
deleted_devices	Статистика видалених пристроїв.
ip_mac_history	Історія змін MAC для IP.
simulated_devices	Емульована топологія мережі.

Після очищення локальний ПК автоматично додається назад у базу, щоб користувач завжди бачив пристрій, з якого запускається сканування.

14. Сценарії тестування працездатності

№	Дія	Очікуваний результат
1	Запустити app.py і відкрити http://127.0.0.1:5000	Відкривається головна сторінка Network Monitor.
2	Перейти на «Сканування» і запустити емуляцію	З'являється спіннер, кнопка блокується, після завершення відкривається список пристроїв.
3	Перевірити сторінку «Пристрої»	Є локальний ПК, роутер та інші емульовані пристрої.
4	Повторити емуляційний скан кілька разів	Більшість пристроїв залишаються, частина може тимчасово зникати. Масового removed після одного скану немає.
5	Натиснути «Дозволити» для unknown-пристрою	Статус змінюється на allowed, у журналі з'являється подія.
6	Натиснути «Заборонити» для unknown-пристрою	Статус змінюється на blocked, у журналі з'являється подія.
7	Натиснути «Видалити» для allowed/blocked-пристрою	Пристрій зникає з основного списку, лічильник видалених збільшується.
8	Перейти до «Журналу попереджень»	Відображаються старт скану, дії користувача та службові події.
9	Перейти до «Історії MAC» після підозрілої події	Відображається запис про зміну MAC для IP.
10	Очистити базу	Дані очищаються, локальний ПК автоматично повертається в список.

15. Типові помилки та їх усунення

Помилка або ситуація	Причина	Рішення
PowerShell блокує activate.ps1	Політика виконання скриптів Windows.	Виконати Set-ExecutionPolicy RemoteSigned для CurrentUser або запускати venv\Scripts\python app.py.
Sniffing and sending packets is not available at layer 2	Не встановлено Npcap/WinPcap.	Встановити Npcap з WinPcap API-compatible mode або використовувати емуляцію.
Порожній список пристроїв після запуску	Сканування ще не виконувалось або база очищена.	Запустити емуляційне або реальне сканування. Локальний ПК має додаватися автоматично.
Один IP має різні MAC	Можлива зміна адаптера, віртуалізація, VPN або ARP spoofing.	Перевірити «Історію MAC» і журнал подій.
Кнопка сканування неактивна	Сканування вже виконується.	Дочекатися завершення операції.
Реальне сканування нічого не знаходить	Немає доступу до мережі, неправильний CIDR, немає прав адміністратора.	Перевірити CIDR, підключення, права адміністратора та Npcap.

16. Обмеження та правила безпечного використання

Програма є навчальним прототипом і не замінює професійні IDS/IPS-рішення.

Реальне сканування слід виконувати лише у власній або дозволеній локальній мережі.

Виявлення IP/MAC-конфлікту не є остаточним доказом атаки; потрібна додаткова перевірка.

Режим емуляції призначений для демонстрації та тестування логіки програми без мережевого обладнання.

У реальному середовищі рекомендується змінити Flask secret_key та не запускати додаток у debug-режимі на відкритому інтерфейсі.

Додаток Б

Лістинг database.py

```
import socket
import uuid
import sqlite3
from datetime import datetime
from config import DATABASE_PATH



def _connect():
    conn = sqlite3.connect(DATABASE_PATH)
    conn.row_factory = sqlite3.Row
    return conn



def _ensure_column(conn, table_name, column_name, column_definition):
    columns = conn.execute(f"PRAGMA
table_info({table_name})").fetchall()
    existing_columns = {column["name"] for column in columns}


    if column_name not in existing_columns:
        conn.execute(f"ALTER TABLE {table_name} ADD COLUMN
{column_name} {column_definition}")


def get_local_device_info(emulation=False):
    hostname = socket.gethostname()


    try:
        ip_address = socket.gethostbyname(hostname)
    except Exception:
        ip_address = "127.0.0.1"


    if emulation and ip_address == "127.0.0.1":
```

```
        ip_address = "192.168.0.100"

    mac_int = uuid.getnode()
    mac_address = ":".join(
        f"{(mac_int >> shift) & 0xff:02X}"
        for shift in range(40, -1, -8)
    )

    return {
        "ip_address": ip_address,
        "mac_address": mac_address,
        "hostname": hostname
    }

def get_emulated_network_cidr():
    local = get_local_device_info(emulation=True)
    ip_parts = local["ip_address"].split(".")

    if len(ip_parts) != 4:
        return "192.168.0.0/24"

    return f"{ip_parts[0]}.{ip_parts[1]}.{ip_parts[2]}.0/24"

def generate_fake_mac(prefix=None):
    import random

    # Для звичайних емульованих пристроїв використовуємо стабільно
реалістичний OUI,
    # а для підозрілих подій можна передати prefix="02".
    if prefix:
        first = prefix.upper()
    else:
        first = random.choice(["A4", "B8", "D4", "E8", "F0", "70",
"88", "9C"])
```

```
    return "%s:%02X:%02X:%02X:%02X:%02X" % (
        first,
        random.randint(0, 255),
        random.randint(0, 255),
        random.randint(0, 255),
        random.randint(0, 255),
        random.randint(0, 255),
    )


def generate_stable_mac(network_prefix, last_octet):
    parts = network_prefix.split(".")
    try:
        b1, b2, b3 = [int(part) & 0xff for part in parts[:3]]
    except Exception:
        b1, b2, b3 = 192, 168, 0

    return f"A4:{b1:02X}:{b2:02X}:{b3:02X}:00:{last_octet & 0xff:02X}"

def ensure_local_device():
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    local = get_local_device_info()
    mac = local["mac_address"]

    with _connect() as conn:
        existing = conn.execute(
            "SELECT * FROM devices WHERE mac_address=?",
            (mac,)
        ).fetchone()

        if existing:
            conn.execute(
                """
                UPDATE devices
```

```
                SET ip_address=?, hostname=?, status=?, last_seen=?,
missed_scans=0
                WHERE mac_address=?
                """,
                (
                    local["ip_address"],
                    local["hostname"] + " (цей ПК)",
                    "allowed",
                    now,
                    mac
                )
            )
        else:
            conn.execute(
                """
                INSERT INTO devices(ip_address, mac_address, hostname,
status, first_seen, last_seen, missed_scans)
                VALUES (?, ?, ?, ?, ?, ?, 0)
                """,
                (
                    local["ip_address"],
                    mac,
                    local["hostname"] + " (цей ПК)",
                    "allowed",
                    now,
                    now
                )
            )

        conn.commit()

    return local

def _network_prefix_from_emulated_ip():
    local = get_local_device_info(emulation=True)
```

```
    ip_parts = local["ip_address"].split(".")

    if len(ip_parts) != 4:
        return "192.168.0", 100

    return ".".join(ip_parts[:3]), int(ip_parts[3])


def _insert_simulated_device(conn, ip_address, mac_address, hostname,
device_type, stability, is_core=False):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    conn.execute(
        """
        INSERT OR IGNORE INTO simulated_devices(
            ip_address, mac_address, hostname, device_type, stability,
is_core, active, created_at, updated_at
        )
        VALUES (?, ?, ?, ?, ?, ?, 1, ?, ?)
        """,
        (ip_address, mac_address.upper(), hostname, device_type,
stability, 1 if is_core else 0, now, now)
    )


def ensure_simulated_topology():
    import random

    local = get_local_device_info(emulation=True)
    network_prefix, local_last_octet =
_network_prefix_from_emulated_ip()
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    with _connect() as conn:
        count = conn.execute("SELECT COUNT(*) AS count FROM
simulated_devices").fetchone()["count"]
```

```
        # Локальний ПК і роутер синхронізуються завжди, навіть якщо топологія вже існує.
        _insert_simulated_device(
            conn, local["ip_address"], local["mac_address"], local["hostname"] + " (цей ПК)",
            "core", 1.0, True
        )
        router_mac = generate_stable_mac(network_prefix, 1)
        backup_router_mac = generate_stable_mac(network_prefix, 2)

        _insert_simulated_device(
            conn, f"{network_prefix}.1", router_mac, "Router",
            "core", 1.0, True
        )

        conn.execute(
            """
            UPDATE simulated_devices
            SET ip_address=?, hostname=?, device_type='core', stability=1.0, is_core=1, active=1, updated_at=?
            WHERE mac_address=?
            """,
            (local["ip_address"], local["hostname"] + " (цей ПК)", now, local["mac_address"].upper())
        )

        # Якщо у старій БД вже були випадково створені дублікати роутера,
        # залишаємо тільки стабільні службові записи.
        conn.execute(
            "DELETE FROM simulated_devices WHERE hostname='Router' AND mac_address != ?",
            (router_mac,)
        )
        conn.execute(
```

```
            "DELETE FROM simulated_devices WHERE hostname='Backup-
Router' AND mac_address != ?",
            (backup_router_mac,)
        )

        if count == 0:
            used_octets = {1, local_last_octet}

            # Додатковий шлюз/резервний маршрутизатор є частиною
топології не завжди.
            if random.random() < 0.55:
                _insert_simulated_device(
                    conn, f"{network_prefix}.2", backup_router_mac,
"Backup-Router",
                    "core", 0.95, True
                )
                used_octets.add(2)

            fixed_devices = [
                ("Printer", "core", 0.92),
                ("NAS", "core", 0.96),
                ("Camera", "core", 0.9),
                ("Workstation", "core", 0.88),
            ]
            mobile_devices = [
                ("Laptop", "mobile", 0.78),
                ("Smartphone", "mobile", 0.68),
                ("Tablet", "mobile", 0.62),
                ("IoT-Device", "mobile", 0.72),
            ]

            initial_devices = random.sample(fixed_devices,
k=random.randint(2, 4))
            initial_devices += random.sample(mobile_devices,
k=random.randint(2, 4))
```

```
            for hostname, device_type, stability in initial_devices:
                last_octet = random.randint(10, 240)
                while last_octet in used_octets:
                    last_octet = random.randint(10, 240)
                used_octets.add(last_octet)

                _insert_simulated_device(
                    conn, f"{network_prefix}.{last_octet}",
generate_fake_mac(),
                    hostname, device_type, stability, False
                )

        conn.commit()


def _add_random_guest_device(conn):
    import random

    network_prefix, local_last_octet =
_network_prefix_from_emulated_ip()
    used_octets = {local_last_octet}
    rows = conn.execute("SELECT ip_address FROM simulated_devices
WHERE active=1").fetchall()

    for row in rows:
        try:
            used_octets.add(int(row["ip_address"].split(".")[-1]))
        except Exception:
            pass

    last_octet = random.randint(20, 250)
    while last_octet in used_octets:
        last_octet = random.randint(20, 250)
```

```
    hostname = random.choice(["Guest-Phone", "Guest-Laptop", "Unknown-
Device", "IoT-Device"])
    _insert_simulated_device(
        conn, f"{network_prefix}.{last_octet}", generate_fake_mac(),
        hostname, "guest", random.uniform(0.35, 0.58), False
    )



def simulate_network_devices():
    """
    Stateful-емуляція локальної мережі.
    Перша емуляція створює базову топологію у SQLite, наступні скани
    використовують її як основу і вносять лише невеликі реалістичні
зміни.
    """
    import random

    ensure_simulated_topology()

    with _connect() as conn:
        # Іноді в мережі з'являється новий гостьовий пристрій.
        active_count = conn.execute(
            "SELECT COUNT(*) AS count FROM simulated_devices WHERE
active=1"
        ).fetchone()["count"]
        if active_count < 12 and random.random() < 0.25:
            _add_random_guest_device(conn)
            conn.commit()

        rows = conn.execute(
            """
            SELECT *
            FROM simulated_devices
            WHERE active=1
            ORDER BY is_core DESC, id ASC
```

```
            """
        ).fetchall()

    devices = []
    optional_devices = []

    for row in rows:
        device = {
            "ip_address": row["ip_address"],
            "mac_address": row["mac_address"],
            "hostname": row["hostname"],
        }

        if row["is_core"]:
            devices.append(device)
        else:
            optional_devices.append((device, float(row["stability"])))

    # Більшість не-core пристроїв залишаються, але частина може
тимчасово зникнути.
    for device, stability in optional_devices:
        if random.random() <= stability:
            devices.append(device)

    # Не менше 4 пристроїв у результаті, щоб скан виглядав змістовно.
    if len(devices) < 4:
        missing = [device for device, _ in optional_devices if device
not in devices]
        random.shuffle(missing)
        devices.extend(missing[: 4 - len(devices)])

    # Обмежуємо розмір одного скану до 10 пристроїв, але core не
викидаємо.
    if len(devices) > 10:
```

```
        core = [d for d in devices if "(цей ПК)" in d["hostname"] or
d["hostname"] in ["Router", "Backup-Router"]]
        non_core = [d for d in devices if d not in core]
        devices = core + random.sample(non_core, k=max(0, 10 -
len(core)))

    # Рідкісна підозріла подія: інший MAC на вже зайнятій IP-адресі.
    candidates = [
        d for d in devices
        if "(цей ПК)" not in d["hostname"] and not
d["ip_address"].endswith(".1")
    ]
    if candidates and random.random() < 0.08:
        target = random.choice(candidates)
        devices.append({
            "ip_address": target["ip_address"],
            "mac_address": generate_fake_mac(prefix="02"),
            "hostname": "Spoofed-Device"
        })

    return devices

def init_db():
    with _connect() as conn:
        conn.execute("""
        CREATE TABLE IF NOT EXISTS devices (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            ip_address TEXT,
            mac_address TEXT UNIQUE NOT NULL,
            hostname TEXT,
            status TEXT NOT NULL DEFAULT 'unknown',
            first_seen TEXT NOT NULL,
            last_seen TEXT NOT NULL
        )
        """)
```

```
conn.execute("""
CREATE TABLE IF NOT EXISTS alerts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    device_mac TEXT NOT NULL,
    device_ip TEXT,
    message TEXT NOT NULL,
    created_at TEXT NOT NULL
)
""")
conn.execute(
    """
    CREATE TABLE IF NOT EXISTS deleted_devices (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        ip_address TEXT,
        mac_address TEXT,
        hostname TEXT,
        deleted_at TEXT
    )
    """
)
conn.execute(
    """
    CREATE TABLE IF NOT EXISTS ip_mac_history (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        ip_address TEXT NOT NULL,
        old_mac_address TEXT,
        new_mac_address TEXT NOT NULL,
        event_type TEXT NOT NULL,
        suspicion_level TEXT NOT NULL,
        message TEXT NOT NULL,
        created_at TEXT NOT NULL
    )
    """
)
```

```
        conn.execute(
            """
            CREATE TABLE IF NOT EXISTS simulated_devices (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                ip_address TEXT NOT NULL,
                mac_address TEXT UNIQUE NOT NULL,
                hostname TEXT NOT NULL,
                device_type TEXT NOT NULL,
                stability REAL NOT NULL DEFAULT 0.8,
                is_core INTEGER NOT NULL DEFAULT 0,
                active INTEGER NOT NULL DEFAULT 1,
                created_at TEXT NOT NULL,
                updated_at TEXT NOT NULL
            )
            """
        )

        # Міграція для вже створених БД: потрібна для відкладеного
видалення.
        _ensure_column(conn, "devices", "missed_scans", "INTEGER NOT
NULL DEFAULT 0")

        conn.commit()

def get_deleted_count():
    with _connect() as conn:
        row = conn.execute(
            "SELECT COUNT(*) AS count FROM deleted_devices"
        ).fetchone()

        return row["count"]

def upsert_scan_result(device):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    mac = device["mac_address"].upper()
```

```
with _connect() as conn:
    existing = conn.execute(
        "SELECT * FROM devices WHERE mac_address = ?",
        (mac,)
    ).fetchone()

    same_ip_other_mac = conn.execute(
        """
        SELECT *
        FROM devices
        WHERE ip_address = ?
        AND mac_address != ?
        AND status != 'removed'
        """,
        (device.get("ip_address"), mac)
    ).fetchone()

    if same_ip_other_mac:
        suspicion_level = calculate_suspicion_level({
            "ip_address": device.get("ip_address"),
            "mac_address": mac,
            "hostname": device.get("hostname", "")
        })

        log_ip_mac_change(
            device.get("ip_address"),
            same_ip_other_mac["mac_address"],
            mac,
            suspicion_level
        )

    if existing:
        if existing["status"] in ["allowed", "blocked"]:
```

```
                new_status = existing["status"]
            else:
                # Якщо пристрій повернувся після тимчасової
відсутності або removed,
                # він знову потребує рішення адміністратора.
                new_status = "unknown"

            conn.execute(
                """
                UPDATE devices
                SET ip_address=?, hostname=?, status=?, last_seen=?,
missed_scans=0
                WHERE mac_address=?
                """,
                (device.get("ip_address"), device.get("hostname", ""),
new_status, now, mac),
            )

            conn.commit()
            return "updated"

        conn.execute(
            """
            INSERT INTO devices(ip_address, mac_address, hostname,
status, first_seen, last_seen, missed_scans)
            VALUES (?, ?, ?, ?, ?, ?, 0)
            """,
            (device.get("ip_address"), mac, device.get("hostname",
""), "unknown", now, now),
        )

        conn.execute(
            """
            INSERT INTO alerts(device_mac, device_ip, message,
created_at)
            VALUES (?, ?, ?, ?)
```

```
            """,
            (mac, device.get("ip_address"), f"Виявлено новий невідомий пристрій: {mac}", now),
        )

        conn.commit()
        return "new_unknown"


def get_devices():
    with _connect() as conn:
        rows = conn.execute("SELECT * FROM devices ORDER BY status DESC, last_seen DESC").fetchall()
        return [dict(r) for r in rows]


def get_alerts(limit=50):
    with _connect() as conn:
        rows = conn.execute("SELECT * FROM alerts ORDER BY created_at DESC LIMIT ?", (limit,)).fetchall()
        return [dict(r) for r in rows]


def approve_device(mac_address):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    mac = mac_address.upper()

    with _connect() as conn:
        device = conn.execute(
            "SELECT * FROM devices WHERE mac_address=?",
            (mac,)
        ).fetchone()

        conn.execute(
            """
```

```
            UPDATE devices
            SET status='allowed'
            WHERE mac_address=?
            """,
            (mac,)
        )

        if device:
            conn.execute(
                """
                INSERT INTO alerts(device_mac, device_ip, message,
created_at)
                VALUES (?, ?, ?, ?)
                """,
                (
                    device["mac_address"],
                    device["ip_address"],
                    f"Пристрій дозволено адміністратором: {mac}",
                    now
                )
            )

        conn.commit()

def delete_device(mac_address):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    mac = mac_address.upper()

    with _connect() as conn:
        device = conn.execute(
            "SELECT * FROM devices WHERE mac_address=?",
            (mac,)
        ).fetchone()
```

```
        if device:
            conn.execute(
                """
                INSERT INTO deleted_devices(ip_address, mac_address,
hostname, deleted_at)
                VALUES (?, ?, ?, ?)
                """,
                (
                    device["ip_address"],
                    device["mac_address"],
                    device["hostname"],
                    now
                )
            )

            conn.execute(
                """
                INSERT INTO alerts(device_mac, device_ip, message,
created_at)
                VALUES (?, ?, ?, ?)
                """,
                (
                    device["mac_address"],
                    device["ip_address"],
                    f"Адміністратор видалив пристрій з бази:
{device['mac_address']} ({device['ip_address']})",
                    now
                )
            )

            conn.execute(
                "DELETE FROM devices WHERE mac_address=?",
                (mac,)
            )
```

```
        else:
            conn.execute(
                """
                INSERT INTO alerts(device_mac, device_ip, message,
created_at)
                VALUES (?, ?, ?, ?)
                """,
                (
                    mac,
                    None,
                    f"Спроба видалення пристрою, якого немає в базі:
{mac}",
                    now
                )
            )

        conn.commit()


def block_device(mac_address):
    mac = mac_address.upper()

    with _connect() as conn:
        conn.execute(
            """
            UPDATE devices
            SET status='blocked'
            WHERE mac_address=?
            """,
            (mac,)
        )

        conn.execute(
            """
```

```
            INSERT INTO alerts(device_mac, device_ip, message,
created_at)
            SELECT mac_address, ip_address, ?, ?
            FROM devices
            WHERE mac_address=?
            """,
            (
                f"Пристрій заборонено адміністратором: {mac}",
                datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
                mac
            )
        )

        conn.commit()

def mark_removed_devices(found_devices, threshold=3):
    """
    Не позначає пристрій removed після одного пропуску.
    Кожен відсутній у поточному скані пристрій накопичує missed_scans.
    removed ставиться тільки після threshold послідовних пропусків.
    """
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    found_macs = [d["mac_address"].upper() for d in found_devices]

    local_mac = get_local_device_info()["mac_address"].upper()

    if local_mac not in found_macs:
        found_macs.append(local_mac)

    with _connect() as conn:
        if found_macs:
            placeholders = ",".join("?" for _ in found_macs)
            rows = conn.execute(
                f"""
```

```
                SELECT * FROM devices
                WHERE mac_address NOT IN ({placeholders})
                AND status != 'removed'
                """,
                found_macs
            ).fetchall()
        else:
            rows = conn.execute(
                "SELECT * FROM devices WHERE status != 'removed'"
            ).fetchall()

        removed_count = 0

        for row in rows:
            missed_scans = int(row["missed_scans"] or 0) + 1

            if missed_scans >= threshold:
                conn.execute(
                    """
                    UPDATE devices
                    SET status='removed', missed_scans=?, last_seen=?
                    WHERE mac_address=?
                    """,
                    (missed_scans, now, row["mac_address"])
                )

                conn.execute(
                    """
                    INSERT INTO alerts(device_mac, device_ip, message,
created_at)
                    VALUES (?, ?, ?, ?)
                    """,
                    (
                        row["mac_address"],
```

```
                        row["ip_address"],
                        f"Пристрій не виявляється {threshold}
сканування підряд і позначений як видалений: {row['mac_address']}",
                        now
                    )
                )

                removed_count += 1
            else:
                conn.execute(
                    """
                    UPDATE devices
                    SET missed_scans=?
                    WHERE mac_address=?
                    """,
                    (missed_scans, row["mac_address"])
                )

                conn.execute(
                    """
                    INSERT INTO alerts(device_mac, device_ip, message,
created_at)
                    VALUES (?, ?, ?, ?)
                    """,
                    (
                        row["mac_address"],
                        row["ip_address"],
                        f"Пристрій тимчасово не виявлено у скані
({missed_scans}/{threshold}): {row['mac_address']}",
                        now
                    )
                )

        conn.commit()
        return removed_count
```

```
def clear_all():
    with _connect() as conn:
        conn.execute("DELETE FROM devices")
        conn.execute("DELETE FROM alerts")
        conn.execute("DELETE FROM deleted_devices")
        conn.execute("DELETE FROM ip_mac_history")
        conn.execute("DELETE FROM simulated_devices")
        conn.commit()

    ensure_local_device()

def add_alert(device_mac="SYSTEM", device_ip="-", message=""):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    if device_mac is None:
        device_mac = "SYSTEM"

    if device_ip is None:
        device_ip = "-"

    with _connect() as conn:
        conn.execute(
            """
            INSERT INTO alerts(device_mac, device_ip, message,
created_at)
            VALUES (?, ?, ?, ?)
            """,
            (device_mac, device_ip, message, now)
        )
        conn.commit()

def calculate_suspicion_level(device):
    hostname = (device.get("hostname") or "").lower()
```

```
    mac = (device.get("mac_address") or "").upper()
    ip = device.get("ip_address") or ""

    score = 0

    # локальний ПК — безпечний
    if "(цей пк)" in hostname:
        return "low"

    # підозрілий MAC (локально згенерований / spoof)
    if mac.startswith("02:"):
        score += 1

    # системні IP (можуть бути шлюзи або підміна)
    if ip.endswith(".1") or ip.endswith(".2"):
        score += 1

    # оцінка
    if score >= 2:
        return "high"
    elif score == 1:
        return "medium"
    else:
        return "low"

def log_ip_mac_change(ip_address, old_mac, new_mac, suspicion_level):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    message = (
        f"Для IP-адреси {ip_address} виявлено зміну MAC-адреси: "
        f"{old_mac} → {new_mac}. Можлива підміна пристрою або ARP
spoofing."
    )
```

```
    emoji = {
        "low": "□",
        "medium": "□",
        "high": "●"
    }.get(suspicion_level, "O")

    with _connect() as conn:
        conn.execute(
            """
            INSERT INTO ip_mac_history(
                ip_address, old_mac_address, new_mac_address,
                event_type, suspicion_level, message, created_at
            )
            VALUES (?, ?, ?, ?, ?, ?, ?)
            """,
            (
                ip_address,
                old_mac,
                new_mac,
                "mac_changed_for_ip",
                suspicion_level,
                message,
                now
            )
        )

        conn.execute(
            """
            INSERT INTO alerts(device_mac, device_ip, message,
created_at)
            VALUES (?, ?, ?, ?)
            """,
            (
                new_mac,
```

```
                ip_address,
                f"{emoji} {message}",
                now
            )
        )

        conn.commit()

def get_ip_mac_history(limit=100):
    with _connect() as conn:
        rows = conn.execute(
            """
            SELECT *
            FROM ip_mac_history
            ORDER BY created_at DESC
            LIMIT ?
            """,
            (limit,)
        ).fetchall()

        return [dict(r) for r in rows]
```

Додаток В

Лістинг app.py

```
from flask import Flask, render_template, request, redirect, url_for,
flash
from database import (
    init_db,
    get_devices,
    get_alerts,
    approve_device,
    block_device,
    delete_device,
    upsert_scan_result,
    clear_all,
    mark_removed_devices,
    get_deleted_count,
    add_alert,
    ensure_local_device,
    get_local_device_info,
    simulate_network_devices,
    get_emulated_network_cidr,
    get_ip_mac_history
)
from scanner import scan_network
from config import DEFAULT_NETWORK_CIDR, DEFAULT_SCAN_MODE


app = Flask(__name__)
app.secret_key =
"b33283080092481534e0c63d65921c452a5c46ece3743b8d9771bf3fd56e432b"


init_db()
ensure_local_device()


def build_cidr_from_local_ip():
    local = get_local_device_info()
```

```
    ip = local["ip_address"]

    if ip == "127.0.0.1":
        ip = "192.168.0.100"

    parts = ip.split(".")

    if len(parts) != 4:
        return DEFAULT_NETWORK_CIDR

    return f"{parts[0]}.{parts[1]}.{parts[2]}.0/24"

@app.route("/")
def index():
    devices = get_devices()
    alerts = get_alerts(limit=10)
    total = len(devices)
    unknown = len([d for d in devices if d["status"] == "unknown"])
    allowed = len([d for d in devices if d["status"] == "allowed"])
    blocked = len([d for d in devices if d["status"] == "blocked"])
    removed = get_deleted_count()

    threat_level = "safe"
    threat_text = "□ Загроз не виявлено"

    if any(a["message"].startswith("●") for a in alerts):
        threat_level = "danger"
        threat_text = "● Виявлено потенційну загрозу"
    elif any(a["message"].startswith("□") for a in alerts):
        threat_level = "warning"
        threat_text = "□ Виявлено підозрілу активність"

    return render_template(
        "index.html",
```

```
        total=total,
        unknown=unknown,
        allowed=allowed,
        removed=removed,
        blocked=blocked,
        alerts=alerts,
        threat_level=threat_level,
        threat_text=threat_text
    )

@app.route("/devices")
def devices():
    return render_template("devices.html", devices=get_devices())

@app.route("/alerts")
def alerts():
    return render_template("alerts.html",
alerts=get_alerts(limit=100))

@app.route("/scan", methods=["GET", "POST"])
def scan():
    if request.method == "POST":
        cidr = request.form.get("cidr", DEFAULT_NETWORK_CIDR).strip()
        mode = request.form.get("mode", DEFAULT_SCAN_MODE)

        if mode != "real":
            cidr = get_emulated_network_cidr()

        local_device = get_local_device_info(emulation=(mode !=
"real"))

        add_alert(
            device_mac=local_device["mac_address"],
            device_ip=local_device["ip_address"],
```

```
            message=f"З локального пристрою запущено нове сканування
мережі. Режим: {mode}, діапазон: {cidr}"
        )
        try:
            if mode == "real":
                found = scan_network(cidr)
            else:
                found = simulate_network_devices()
            new_unknown = 0

            for item in found:
                result = upsert_scan_result(item)
                if result == "new_unknown":
                    new_unknown += 1

            removed_count = mark_removed_devices(found)

            flash(
                f"Сканування завершено. "
                f"Знайдено пристроїв: {len(found)}. "
                f"Нових невідомих: {new_unknown}. "
                f"Видалених/відсутніх: {removed_count}."
            )

            alerts = get_alerts(limit=20)

            high = any(a["message"].startswith("●") for a in alerts)
            medium = any(a["message"].startswith("□") for a in alerts)

            if high:
                flash("🚨 Виявлено потенційно небезпечну активність у
мережі!")
            elif medium:
                flash("⚠ Виявлено підозрілу активність у мережі.")
```

```
            else:
                flash("✅ Загроз не виявлено.")
        except Exception as exc:
            flash(f"Помилка сканування: {exc}")
            add_alert(
                message=f"Помилка сканування: {exc}"
            )
        return redirect(url_for("devices"))

    return render_template("scan.html",
default_cidr=build_cidr_from_local_ip())


@app.route("/history")
def history():
    return render_template("history.html",
history=get_ip_mac_history(limit=100))


@app.route("/approve/<mac_address>", methods=["POST"])
def approve(mac_address):
    approve_device(mac_address)
    flash("Пристрій позначено як дозволений.")
    return redirect(url_for("devices"))


@app.route("/block/<mac_address>", methods=["POST"])
def block(mac_address):
    block_device(mac_address)
    flash("Пристрій позначено як заборонений.")
    return redirect(url_for("devices"))


@app.route("/delete/<mac_address>", methods=["POST"])
def delete(mac_address):
    delete_device(mac_address)
    flash("Пристрій позначено як видалений.")
    return redirect(url_for("devices"))
```

```
@app.route("/clear", methods=["POST"])
def clear():
    clear_all()
    flash("Базу очищено.")
    return redirect(url_for("index"))

if __name__ == "__main__":
    app.run(host="127.0.0.1", port=5000, debug=True)
```

Додаток Г

Лістинг scanner.py

```
import random
import socket



def _hostname_by_ip(ip_address):
    try:
        return socket.gethostbyaddr(ip_address)[0]
    except Exception:
        return "unknown-host"



def scan_network(cidr):
    """Real ARP scan. Requires scapy and administrator/root
privileges."""
    try:
        from scapy.all import ARP, Ether, srp
    except ImportError as exc:
        raise RuntimeError("Для реального сканування встановіть scapy:
pip install scapy") from exc

    arp = ARP(pdst=cidr)
    ether = Ether(dst="ff:ff:ff:ff:ff:ff")
    packet = ether / arp
    answered, _ = srp(packet, timeout=2, verbose=False)

    devices = []
    for _, received in answered:
        ip = received.psrc
        devices.append({
            "ip_address": ip,
            "mac_address": received.hwsrc,
            "hostname": _hostname_by_ip(ip),
```

```
        })
    return devices



def simulate_scan():
    """Offline emulation for a PC without network access."""
    base_devices = [
        {"ip_address": "192.168.1.1", "mac_address":
"AA:BB:CC:00:00:01", "hostname": "router-local"},
        {"ip_address": "192.168.1.10", "mac_address":
"AA:BB:CC:00:00:10", "hostname": "admin-pc"},
        {"ip_address": "192.168.1.21", "mac_address":
"AA:BB:CC:00:00:21", "hostname": "printer"},
    ]
    # Sometimes add a suspicious new device, so alerts can be tested
repeatedly.
    if random.random() > 0.35:
        suffix = random.randint(30, 99)
        base_devices.append({
            "ip_address": f"192.168.1.{suffix}",
            "mac_address": f"DE:AD:BE:EF:00:{suffix:02X}",
            "hostname": "unknown-device",
        })
    return base_devices
```