Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Бойцуна Дмитра Олеговича

на здобуття ступеня вищої освіти Бакалавра

Система виявлення та протидії ботнет-атакам на основі мультиагентного підходу

Галузь знань 12 - Інформаційні технології
Спеціальність 125 - Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ.220235.22.02.23 ПЗ

Виконав студент 4 курсу група КБ-22-2 Дмитро БОЙЦУН
Керівник д-р. філософії Микола СТЕЦЮК
Нормоконтролер д-р філософії Наталія ПЕТЛЯК

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

11.06 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

09 лютого 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бойцуну Дмитру Олеговичу

1 Тема роботи Система виявлення та протидії ботнет-атакам на основі мультиагентного підходу.

Керівник роботи д-р філософії, ст. викладач Микола Стецюк

Затверджено наказом ректора університету від 8 січня 2026 № 11

2 Строк подання студентом кваліфікаційної роботи на кафедру 25 травня 2026

3 Вихідні дані до роботи Проаналізувати предметну область кіберзахисту від ботнет-атак та обмеження існуючих систем виявлення вторгнень. Сформулювати постановку задачі та визначити функціональні вимоги до системи. Розробити гібридну мультиагентну архітектуру з керуючим вузлом і автономними агентами. Обгрунтувати вибір технологічного стеку. Реалізувати керуючий вузол з підсистемою виявлення на основі правил у форматі YAML. Реалізувати автономних агентів з підтримкою режиму роботи в умовах ізоляції. Розробити механізм накопиченої пам'яті взаємодій із джерелами загроз. Провести тестування системи на реальних атаках та виконати порівняльний аналіз з існуючими рішеннями.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ. Аналіз предметної області та існуючих систем виявлення вторгнень. Постановка задачі. Проєктування гібридної мультиагентної архітектури. Розробка керуючого вузла та підсистеми виявлення на основі YAML-правил. Розробка автономних агентів з підтримкою режиму ізоляції. Реалізація механізму накопиної пам'яті загроз. Тестування виявлення та оцінка показників ефективності. Порівняння з існуючими рішеннями. Висновки.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
«Гібридна архітектура мультиагентної системи SENTARA». «Життєвий цикл взаємодії з джерелом загрози». «Алгоритм роботи агента в автономному режимі».

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7 Дата видачі завдання 09 лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	
Ознайомлення з предметною областю	Лютий	
Дослідження існуючих рішень	Лютий	
Постановка задачі	Березень	
Визначення загальних принципів рішення задачі	Березень	
Деталізація принципів рішення задачі	Квітень	
Розробка проєктних рішень	Квітень	
Апробація проєктних рішень	Травень	
Оформлення пояснювальної записки згідно вимог	Травень	
Оформлення графічної частини	Травень	
Захист КР	Червень	

Студент _____ Дмитро БОЙЦУН

Керівник кваліфікаційної роботи _____ Микола СТЕЦЮК

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система виявлення та протидії ботнет-атакам на основі мультиагентного підходу

Автор роботи: Бойцун Дмитро Олегович.

Керівник роботи: Стецюк Микола Васильович.

Пояснювальна записка: 72 с., 3 додатки, 11 рисунків, 13 таблиць, 42 джерела.

Графічна частина: 3 креслення.

Ключові слова: мультиагентна система, виявлення вторгнень, ботнет, автономний захист, кумулятивна пам'ять загроз.

Кваліфікаційна робота бакалавра присвячена розробці мультиагентної системи виявлення та протидії ботнет-атакам.

У роботі проведено аналіз сучасного стану ботнет-загроз та архітектурних обмежень існуючих систем виявлення вторгнень (Snort, Suricata, Zeek, OSSEC/Wazuh). Досліджено мультиагентний підхід як перспективний напрям подолання залежності захисних механізмів від центрального вузла та реактивного характеру реагування на загрози. Розроблено гібридну архітектуру системи з керуючим вузлом і автономними агентами, здатними самостійно виявляти та блокувати загрози в умовах ізоляції від центру керування. Реалізовано серверну частину системи з використанням Python, FastAPI та PostgreSQL із декларативною мовою опису правил у форматі YAML і механізмом накопиченої пам'яті взаємодій із джерелами загроз. Проведено тестування системи на реальних спробах несанкціонованого доступу, що підтвердило її ефективність, відмовостійкість та можливість застосування у сучасних інфраструктурах кібербезпеки.

25.05.2026 

ABSTRACT

Subject of qualification work: Multi-agent system for detection and botnet attack mitigation.

Author: Boitsun Dmytro Olehovich.

Head of work: Stetsiuk Mykola Vasylovych.

Explanatory note: 72 p., 3 appendices, 11 figures, 13 tables, 42 sources

Graphic part: 3 drawings

Keywords: multi-agent system, intrusion detection, botnet, autonomous defense, cumulative threat memory.

The bachelor's qualification work is devoted to the development of a hybrid multi-agent system for detection and botnet attack mitigation.

The work analyzes the current state of botnet threats and the architectural limitations of existing intrusion detection systems (Snort, Suricata, Zeek, OSSEC/Wazuh). The multi-agent approach is studied as a promising direction for overcoming the dependence of defense mechanisms on a central node and the reactive nature of threat response. A hybrid system architecture with a central control node and autonomous agents capable of independently detecting and blocking threats under conditions of isolation from the control center has been developed. The server side of the system is implemented using Python, FastAPI, and PostgreSQL, with a declarative rule description language in YAML format and a mechanism for accumulated memory of interactions with threat sources. Testing of the system on real unauthorized access attempts confirmed its effectiveness, fault tolerance, and applicability in modern cybersecurity infrastructures.

25.05.2026

ЗМІСТ

ВСТУП

Ботнет-атаки залишаються одним із найбільш небезпечних інструментів сучасної кіберзлочинності. За даними CERT-UA, у 2026 році в Україні зафіксовано 4 315 кіберінцидентів - на 69,8% більше порівняно з попереднім роком; значна частина із них пов'язана з атаками типу «відмова в обслуговуванні» (DDoS), що здійснюються через ботнети. У глобальному масштабі щодоби фіксується активність понад 12 мільйонів унікальних ботів (Spamhaus Project, 2024), а середня вартість інциденту інформаційної безпеки для організації, за оцінкою IBM (2026), складає 3,31 млн доларів США. Сучасні ботнети четвертого покоління використовують однорангову (P2P) архітектуру, шифрований канал командно-керуючого зв'язку (Command-and-Control, C2) на основі протоколів TLS та DNS-over-HTTPS, а також модульну структуру, що суттєво ускладнює їх виявлення традиційними засобами.

Наявні системи виявлення та запобігання вторгненням (IDS/IPS) - Snort, Suricata, Zeek, OSSEC/Wazuh - мають низку системних обмежень при протидії розподіленим ботнет-атакам. До них належать: відсутність накопиченої історії взаємодій із джерелами загроз, через що кожна подія аналізується ізольовано від попередніх; жорстка залежність агентів від центрального сервера, за якої втрата зв'язку призводить до повної втрати функцій виявлення на захищених вузлах; переважно реактивний характер реагування - виявлення відбувається постфактум. Перспективним напрямом для подолання цих обмежень є мультиагентний підхід, за якого автономні агенти здатні самостійно виявляти, класифікувати та блокувати загрози навіть в умовах ізоляції від керуючого вузла.

Мета роботи - підвищення ефективності протидії ботнет-атакам шляхом проєктування та реалізації мультиагентної системи виявлення й блокування загроз, що поєднує накопичену історію взаємодій із джерелами, автономність агентів і централізовану кореляцію подій.

Для досягнення мети поставлено наступні задачі:

1. Провести аналіз сучасного стану ботнет-загроз, класифікувати методи виявлення та визначити обмеження існуючих IDS/IPS.

2. Дослідити теоретичні засади мультиагентного підходу до кіберзахисту й обґрунтувати вибір гібридної архітектури типу «керуючий вузол - агент».

3. Проаналізувати поточний стан протидії кіберзагрозам на підприємстві, виявити системні недоліки та сформулювати вимоги до системи.

4. Спроєктувати та реалізувати мультиагентну систему SENTARA: керуючий вузол на основі фреймворку FastAPI, СКБД PostgreSQL та сховища Redis з модулем виявлення на правилах у форматі YAML; автономні агенти на основі systemd, nftables та SQLite з підтримкою функціонування в режимі ізоляції від керуючого вузла; підсистему накопичення історії загроз з ієрархічною класифікацією джерел за рівнем ворожості.

5. Реалізувати підсистему збереження стану системи (SEED), що забезпечує її відновлення після втрати керуючого вузла, та інтерфейс взаємодії з оператором на основі Telegram з модулем адаптивного формування сповіщень.

6. Провести оцінку ефективності розробленої системи: порівняльний аналіз із Snort, Suricata, Zeek і Wazuh; експериментальне тестування процесів виявлення загроз та роботи в автономному режимі; техніко-економічний аналіз вартості впровадження.

Об'єкт дослідження - процес виявлення та протидії ботнет-атакам у розподілених комп'ютерних мережах.

Предмет дослідження - мультиагентна система протидії ботнет-атакам з кумулятивною пам'яттю загроз та автономними агентами.

Методи дослідження

У роботі використано: системний аналіз - для класифікації ботнетів і методів їх виявлення; порівняльний аналіз - для оцінки наявних рішень класу IDS/IPS; методи моделювання - для проєктування архітектури «керуючий вузол - агент»; експериментальне дослідження - для оцінки показників виявлення, функціонування в автономному режимі та продуктивності системи на стенді з трьох вузлів; техніко-економічний аналіз - для зіставлення вартості впровадження з комерційними та відкритими SIEM-рішеннями.

Практичне значення одержаних результатів полягає у створенні працездатної системи SENTARA, що забезпечує: середній час виявлення загрози

(MTTD, Mean Time To Detect) не більше 8 секунд для атак методом перебору паролів служби SSH, що понад у 450 разів швидше за ручний аналіз журналів; автоматичне блокування джерела загрози засобами nftables менш ніж за 5 мілісекунд; автономне функціонування агента протягом понад 72 годин у разі втрати зв'язку з керуючим вузлом; повноту виявлення (Detection Rate) 100% для шести категорій тестових атак; ієрархічну класифікацію джерел загроз за чотирма рівнями ворожості - від невідомого до постійного порушника. Орієнтовна вартість впровадження складає 20–30 євро на місяць, що у 2–150 разів менше за вартість зіставних рішень. Розроблену систему може бути адаптовано для використання у підрозділах кіберполіції, центрах операційного моніторингу безпеки (SOC) та на підприємствах з розподіленою інформаційною інфраструктурою.

Уперше запропоновано модель накопиченої історії взаємодій із джерелами загроз, що, на відміну від наявних систем виявлення вторгнень, зберігає повну хронологію подій для кожного джерела та використовує її для ієрархічної класифікації за чотирма рівнями ворожості (невідоме джерело, разовий порушник, повторний порушник, постійний порушник) із подальшим адаптивним реагуванням;

Набув подальшого розвитку підхід до забезпечення відмовостійкості агентів розподілених систем виявлення вторгнень: запропоновано механізм автономного функціонування агента з подальшою подієво-орієнтованою синхронізацією подій (event sourcing), що забезпечує безперервне виконання захисних функцій протягом не менше 72 годин у разі втрати зв'язку з керуючим вузлом, з гарантованим відновленням повної послідовності накопичених подій після відновлення з'єднання.

Розроблену систему протестовано на лабораторному стенді з трьох вузлів із використанням стандартних інструментів тестування на проникнення (hydra, nmap, sqlmap). За результатами випробувань повнота виявлення атак склала 100% для шести категорій загроз, F1-міра - 95,9% за 48 годин безперервного тестування. Тривалість функціонування агента в умовах ізоляції від керуючого вузла становила 4 години, час подальшої синхронізації накопичених подій - 1,8 секунди.

Робота складається зі вступу, трьох розділів, висновків та переліку джерел

посилань. У першому розділі проведено аналіз ботнет-загроз і методів їх виявлення, розглянуто теоретичні засади мультиагентних систем, оцінено поточний стан захисту інформаційної інфраструктури підприємства та сформульовано вимоги до системи, що розробляється. Другий розділ присвячено архітектурі та програмній реалізації сиснеми SENTARA: гібридній моделі «керуючий вузол - агент», модулю виявлення на правилах у форматі YAML, підсистемі накопичення історії загроз, автономним агентам з підтримкою режиму ізоляції, а також механізму збереження та відновлення стану системи. У третьому розділі наведено результати оцінки ефективності: порівняльний аналіз з наявними рішеннями класу IDS, експериментальне тестування, техніко-економічне обґрунтування та практичні рекомендації щодо впровадження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ПРОТИДІЇ БОТНЕТ-АТАКАМ

1.1Аналіз сучасного стану загроз ботнет-атак

Ботнет (botnet, від robot + network) – це мережа скомпрометованих комп'ютерних пристроїв (ботів), об'єднаних під централізованим або децентралізованим управлінням зловмисника (ботмастера) для виконання шкідливих дій без відома власників цих пристроїв. Ботнети є одним із найбільш небезпечних інструментів кіберзлочинності: за даними CERT-UA, у 2026 році в Україні зафіксовано 4 315 кіберінцидентів, що на 69,8% більше порівняно з попереднім роком, значна частина яких пов'язана з DDoS-атаками, що здійснюються через ботнети [1].

Еволюція ботнетів пройшла кілька поколінь. Перше покоління (2000-2005) використовувало централізовану архітектуру Command-and-Control (C2) на базі IRC-каналів: ботмастер надсилав команди через IRC-сервер, а всі боти підключалися до одного каналу для отримання інструкцій. Ця архітектура мала критичну вразливість - знищення IRC-сервера виводило з ладу весь ботнет.

Друге покоління (2005-2012) перейшло на HTTP-базовані C2-сервери, що дозволило маскувати C2-трафік під легітимний веб-трафік і ускладнило виявлення. Ботнети Zeus, SpyEye та Citadel використовували HTTP POST-запити до веб-панелей для отримання команд та ексфільтрації вкрадених даних (банківські облікові дані, паролі).

Третє покоління (2012-2020) впровадило децентралізовану peer-to-peer (P2P) архітектуру, де боти обмінюються командами безпосередньо між собою без єдиного C2-сервера. Ботнети Mirai (2016), GameOver Zeus та ZeroAccess використовували P2P для забезпечення стійкості: знищення частини вузлів не впливає на працездатність решти мережі.

Четверте покоління (2020-сьогодні) характеризується використанням IoT-пристроїв (камери, роутери, розумні побутові прилади), шифрованих протоколів (TLS, DNS-over-HTTPS) для маскування C2-комунікації та модульної архітектури, де бот може динамічно завантажувати модулі для різних типів атак.

Ботнет Emotet, розбитий у 2021 році та відновлений у 2022, є показовим прикладом: він використовував спам-модуль для розповсюдження, банківський троян для крадіжки коштів та модуль ransomware-as-a-service для шифрування данних жертв [2, 3].

Для систематизації еволюції проведено порівняльний аналіз найвідоміших ботнетів кожного покоління (таблиця 1.1).

Таблиця 1.1 - Порівняльний аналіз найвідоміших ботнетів за поколіннями

Ботнет	Рік	Покоління	Архітектура C2	Пік ботів	Тип атак	Статус
EarthLink Spammer	2000	1 (IRC)	Централізована IRC	~25 000	Спам	Знищено 2000
Storm	2007	2 (HTTP+P2P)	Гібридна HTTP + Overnet P2P	~1 000 000	Спам, DDoS	Знищено 2008
Zeus/Zbot	2007	2 (HTTP)	Централізована HTTP панель	~3 600 000	Банківська крадіжка	Код витік 2011
Conficker	2008	2 (HTTP)	Централізована + DGA	~10 000 000	Malware delivery	Залишки активні
Mirai	2016	3 (IoT)	Централізована + сканування	~600 000 IoT	DDoS 1.2 Тбіт/с	Код відкритий, варіанти активні
Necurs	2012	3 (P2P)	P2P + DGA	~9 000 000	Спам 12 млн/год, ransomware	Знищено 2020
Emotet	2014	4 (модульний)	Багаторівнева HTTP + TLS	~1 500 000	Модульний: спам, banking, RaaS	Знищено 2021, відновлено 2022
Mozi	2019	4 (IoT P2P)	DHT-based P2P	~1 500 000 IoT	DDoS, криптомайнінг	Kill switch 2023

Аналіз таблиці демонструє чіткий тренд: кожне покоління стає стійкішим до знищення. Централізовані IRC-ботнети знищувалися за місяці, а сучасні P2P IoT-ботнети (Mirai, Mozi) продовжують функціонувати роками навіть після

часткових takedown-операцій, оскільки кожен бот є одночасно і клієнтом, і сервером. Ботнет Mirai, вихідний код якого опублікований у 2016 році, породив десятки варіантів, що активні дотепер (рисунок 1.1).

Окремої уваги заслуговує тенденція до модульності. Emotet, який початково був банківським трояном, еволюціонував у платформу "malware-as-a-service": ботмастер динамічно завантажує на заражені пристрої різні модулі - від спам-розсилки до ransomware - залежно від цілей. Це ускладнює виявлення, оскільки поведінка бота може кардинально змінитися протягом годин без оновлення основного коду [2, 3].

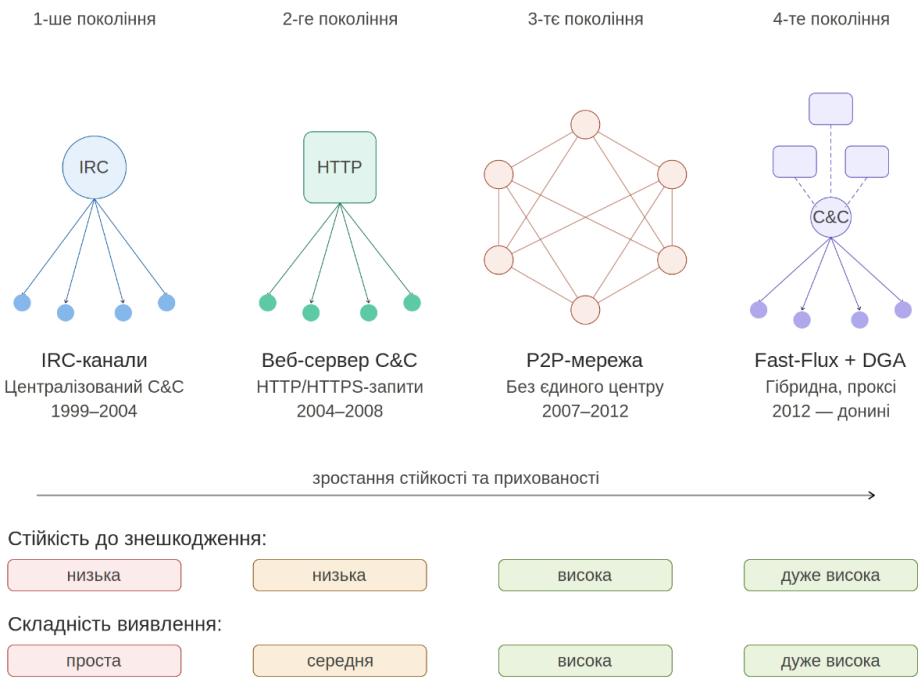


Рисунок 1.1 – Еволюція архітектур ботнетів

Ботнети використовуються для широкого спектру кіберзлочинних дій. DDoS-атаки (Distributed Denial of Service) - найпоширеніший тип: тисячі ботів одночасно надсилають запити до цільового сервера, виснажуючи ресурси. За даними Cloudflare Radar (2026), середня потужність DDoS-атак зросла до 1,2 Тбіт/с, а рекордна атака досягла 5,6 Тбіт/с. Спам та фішинг - боти масово розсилають електронні листи зі шкідливими посиланнями; Necurs у пік розсилав

12 млн спам-повідомлень/год. Крадіжка облікових даних - кейлогери та формграббери перехоплюють паролі, банківські реквізити та cookie-файли. Криптоджекінг - несанкціоноване використання ресурсів жертви для майнінгу; Smominru заразив 500 000+ серверів і добував Monero на ~$3,6 млн. Ransomware delivery - ботнети як канал доставки програм-вимагачів (Emotet → Ryuk, Trickbot → Conti) [4, 5].

Україна, перебуваючи в умовах збройного конфлікту, є одним із найбільш атакованих кіберпросторів. За даними Держспецзв'язку та CERT-UA, у 2024 році зафіксовано 4 315 кіберінцидентів (зростання 69,8% до 2023), з них: критичних - 1 042, високого рівня - 1 799, середнього - 1 474. Значна частина атак здійснювалася через ботнети, контрольовані угрупованнями, пов'язаними з РФ (Sandworm, Fancy Bear, Gamaredon) [1].

У глобальному масштабі, за даними Spamhaus Project (2024), щоденно активні 12 млн+ унікальних IP-адрес ботів. Nokia Threat Intelligence Report (2024) зазначає: IoT-пристрої складають 40% усіх ботнет-вузлів. Середня вартість DDoS-атаки на чорному ринку - від $50 (10 хв, 10 Гбіт/с) до $10 000 (24 год, 1 Тбіт/с), що робить ботнет-атаки доступними навіть для зловмисників з мінімальним бюджетом. Середня вартість інциденту безпеки для організацій з < 500 співробітниками складає $3.31 млн (IBM, 2024), при цьому середній час перебування зловмисника у системі до виявлення - 204 дні для організацій без SIEM та 73 дні - з ним [26, 27, 28].

Методи виявлення ботнетів поділяються на чотири категорії.

Сигнатурний аналіз (signature-based) - порівняння трафіку з базою відомих зразків. Переваги: precision > 99% для відомих загроз, низький FP rate. Недоліки: нездатність виявляти zero-day, необхідність постійного оновлення бази, вразливість до поліморфізму.

Аналіз аномалій (anomaly-based) - побудова статистичної моделі "нормальної" поведінки та виявлення відхилень. Переваги: виявлення невідомих ботнетів та zero-day. Недоліки: FP rate 5-30%, тривалий baseline period, складність налаштування порогів.

Аналіз мережевих потоків (flow-based) - дослідження метаданих з'єднань

(NetFlow/IPFIX) без аналізу вмісту. Ефективний для виявлення C2: періодичні підключення, аномальні DNS-запити, beaconing (регулярні "пульсації" з фіксованим інтервалом).

Поведінковий аналіз на рівні хоста (host-based) - моніторинг процесів, файлової системи, системних викликів. Ефективний для постексплуатаційної активності: нові процеси, зміна автозавантаження, шифрування файлів, lateral movement.

Обмеження існуючих систем виявлення

Snort (Cisco/Open Source, GPLv2) - найпоширеніша мережева IDS з 35 000+ сигнатур. Обмеження: переважно сигнатурний підхід (нові ботнети невидимі до оновлення), single-node без кластеризації, високий CPU при > 1 Гбіт/с.

Suricata (OISF, GPLv2) - багатопоточна IDS/IPS з GPU-прискоренням. Переваги: JA3/JA4 TLS fingerprinting для C2 через HTTPS. Обмеження: потребує 8+ ядер / 16+ ГБ RAM, відсутність агентного підходу.

Zeek (раніше Bro, BSD) - фреймворк мережевого аналізу з власною мовою скриптування. Переваги: глибокий аналіз протоколів, поведінковий аналіз. Обмеження: складна мова (високий поріг), лише мережевий трафік (не хостові події).

OSSEC/Wazuh (GPLv2) - хостова IDS з агентами. Переваги: FIM, аналіз логів, розподілена архітектура. Обмеження: реактивний підхід, відсутність поведінкового аналізу для C2-beaconing, агенти залежні від Manager (при його втраті - лише збір логів, без аналізу та блокування).

Для кількісного порівняння проведено бальну оцінку за 10 критеріями протидії ботнетам (таблиця 1.2).

Максимум серед існуючих - 10/30 (Wazuh, 33%). Жодна система не набрала балів у 4 категоріях: автономність агентів, кумулятивна пам'ять, event sourcing, YAML-правила. Ці 12 балів (40% максимуму) визначають gap, що має заповнити SENTARA. Snort та Suricata - мережеві, не бачать хостових подій. Zeek потужний аналітично, але без хостів та з високим порогом входження. Wazuh найближчий архітектурно, але агенти без автономного Detection Engine при втраті Manager.

Таблиця 1.2 – Бальна оцінка IDS/IPS за критеріями протидії ботнетам (0-3 бали)

Критерій	Snort	Suricata	Zeek	Wazuh	Max
Сигнатурне виявлення	3	3	2	2	3
Поведінковий аналіз	0	1	3	1	3
C2 traffic detection (TLS)	0	2	2	0	3
Розподілений моніторинг	0	0	2	3	3
Автономність агентів	0	0	0	0	3
Автоблокування (IPS)	1	3	0	1	3
Кумулятивна пам'ять загроз	0	0	0	0	3
Event sourcing / sync	0	0	0	0	3
Хостовий моніторинг	0	0	0	3	3
YAML / декларативні правила	0	0	0	0	3
Разом	4/30	9/30	9/30	10/30	30

1.2 Мультиагентний підхід до виявлення кіберзагроз

Теоретичні основи мультиагентних систем

Мультиагентна система (Multi-Agent System, MAS) - розподілена обчислювальна система з множини автономних агентів, що взаємодіють для досягнення спільної мети. Кожен агент має: автономність (дії без втручання людини), реактивність (реакція на зміни середовища), проактивність (ініціювання дій для цілі) та соціальність (взаємодія з іншими агентами) [6].

У кібербезпеці MAS має три фундаментальні переваги. Розподіленість: агенти розміщені на захищених вузлах і збирають інформацію локально без передачі всього трафіку на центральний сервер - зменшує навантаження та дозволяє аналізувати хостові події. Автономність: при втраті зв'язку агент продовжує захист на локальних правилах. Масштабованість: додавання вузла = розгортання агента без зміни архітектури [7].

Архітектурні моделі MAS для кіберзахисту

Три основні моделі. Централізована (hub-and-spoke): агенти надсилають події на сервер для аналізу. Приклад: Wazuh. Переваги: простота, централізована

кореляція. Недоліки: SPOF, затримка, обмежена масштабованість. Повністю децентралізована (P2P): рівноправні агенти, колективні рішення. Переваги: без SPOF. Недоліки: складність координації, Byzantine fault tolerance, високі вимоги до bandwidth. Гібридна (hierarchical): центральний сервер координує + агенти автономні локально. Поєднує переваги обох: централізована аналітика + автономний захист при втраті зв'язку. Саме гібридна модель обрана для SENTARA (рисунок 1.2).

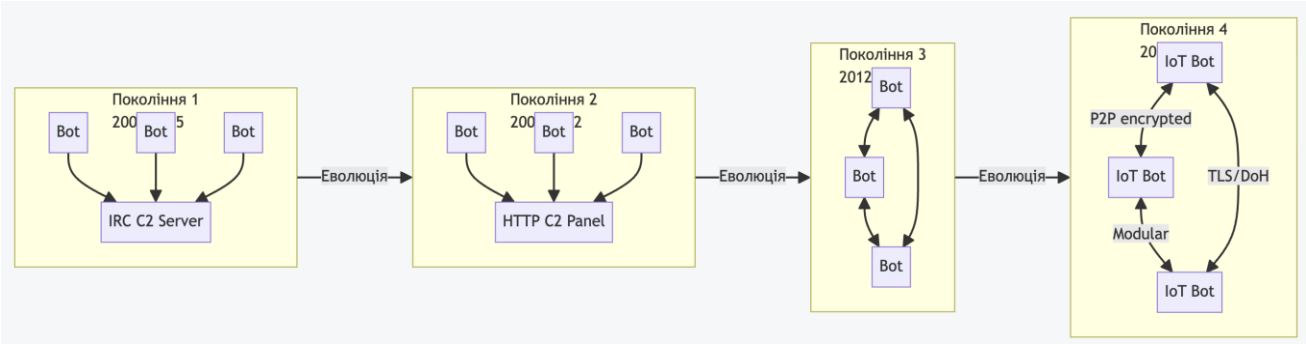


Рисунок 1.2 – Порівняння архітектурних моделей MAS (див. окремо)

Ключові вимоги до мультиагентної системи протидії ботнетам

На підставі проведеного аналізу ботнет-загроз (підрозділ 1.1) та порівняння архітектурних моделей мультиагентних систем (п. 1.2.2) сформульовано п'ять ключових вимог до системи, що розробляється:

Система повинна мати гібридну архітектуру типу «керуючий вузол - агент», за якої агент здатний продовжувати виконання захисних функцій у разі втрати зв'язку з керуючим вузлом. Така властивість необхідна для протидії атакам, що супроводжуються тимчасовим виведенням з ладу центральних компонентів інфраструктури.

Виявлення загроз має бути багаторівневим і поєднувати три підходи: сигнатурний аналіз для відомих типів атак, поведінковий аналіз для виявлення відхилень від штатної активності та кореляційний аналіз для розпізнавання розподілених атак, що залишаються непомітними на рівні окремого вузла.

Система повинна забезпечувати накопичення історії взаємодій з кожним джерелом загрози. На відміну від традиційного ізольованого аналізу окремих

подій, такий підхід дає можливість приймати рішення щодо реагування з урахуванням попередньої активності джерела.

Реагування на виявлені загрози має виконуватися автоматично - шляхом блокування джерела засобами пакетного фільтра nftables без необхідності підтвердження оператором. Це усуває часовий розрив між виявленням і блокуванням, який є критичним для атак з високою швидкістю розгортання.

Обмін даними між агентом і керуючим вузлом має ґрунтуватися на принципі подієво-орієнтованого зберігання (event sourcing), за якого предметом передачі є послідовність подій, а не поточний стан. Такий підхід забезпечує детерміновану синхронізацію після відновлення зв'язку та повну ретроспективну простежуваність дій системи.

1.3 Аналіз поточного стану протидії кіберзагрозам на підприємстві

Характеристика підприємства та його інфраструктури

Базою практики є Департамент кіберполіції Національної поліції України (Хмельницька область). Підрозділ забезпечує протидію кіберзлочинності: розслідування кіберінцидентів, аналіз шкідливого ПЗ, моніторинг інтернет-ресурсів та координація з CERT-UA [1].

Департамент виконує оперативні функції: моніторинг інтернет-ресурсів на шахрайство та розповсюдження malware, розслідування інцидентів (фішинг, несанкціонований доступ, крадіжка даних), аналіз зразків шкідливого ПЗ у sandbox, координація з CERT-UA та правоохоронними органами інших держав, профілактична робота (навчання кібергігієні).

Інфраструктура: робочі станції аналітиків (Windows 10/11, Kali Linux), сервери аналізу (Ubuntu 22.04 LTS, 8 vCPU, 32 ГБ RAM для sandbox та Suricata), мережеве обладнання (Cisco ISR 4000, Mikrotik RB4011), зовнішній канал 1 Гбіт/с (Укртелеком). Загальна кількість вузлів, що потребують захисту - 15 (5 серверів + 10 робочих станцій).

Особливість інфраструктури кіберполіції - подвійна роль: вузли одночасно

є об'єктами захисту та інструментами розслідування (сканування підозрілих IP, завантаження зразків malware може провокувати зворотні атаки). Це підвищує вимоги до системи: потрібно не лише виявляти атаки, але й відрізняти легітимну розслідувальну активність від шкідливої.

Поточний стан: мережевий аналіз - Wireshark (ручний аналіз pcap), сигнатурне виявлення - Suricata на одному вузлі, аналіз логів - ручний через journalctl та grep. Відсутні: централізований збір подій, автоблокування, кумулятивна пам'ять, кореляція між вузлами.

Виявлені недоліки поточного підходу:

1. Відсутність розподіленого моніторингу: Suricata на одному вузлі. Боти, розподілені по кількох вузлах, залишаються невиявленими через відсутність кореляції;

2. Реактивний підхід: виявлення постфактум через ручний аналіз. MTTD > 1 години (ручний) vs секунди (автоматизований);

3. Відсутність автоблокування: після виявлення - ручне блокування через iptables/nftables. Вікно вразливості між виявленням та блокуванням;

4. Відсутність пам'яті загроз: інформація не систематизована. Повторна атака з тієї ж IP аналізується з нуля;

5. SPOF: Suricata на одному вузлі - єдина точка відмови. Збій = повна втрата моніторингу;

6. Обмежений аналіз шифрованого трафіку: сучасні ботнети використовують TLS для C2. Suricata без SSL inspection бачить лише метадані (JA3, SNI).

1.4 Оцінка потенційних збитків від ботнет-атак

За даними IBM (2026): середня вартість інциденту $3.31 млн, кожна хвилина скорочення MTTD зменшує вартість на $1 590 (Ponemon Institute). При MTTD = 8 с (SENTARA) vs > 60 хв (ручний): різниця ~60 хв. Для 10 інцидентів/рік: (60 хв) × $1 590 × 10 = ~$954 000/рік потенційно відвернутих збитків. Навіть при

консервативних 10% = $95 200/рік - інвестиції €240-360/рік окупаються у 250+ разів [26, 34].

Для кіберполіції додатково: скорочення часу розслідування (кумулятивна пам’ять з повною історією), зменшення навантаження на аналітиків (автоблокування замість ручного), та покращення якості доказів (event sourcing - повна хронологія для суду).

Окремо варто оцінити вартість людино-годин. Аналітик кіберполіції витрачає в середньому 2-3 години на ручний аналіз одного інциденту SSH brute force: перегляд auth.log (30 хв), ідентифікація IP (15 хв), перевірка по базах AbuseIPDB (15 хв), створення правила nftables (15 хв), документування (30-60 хв). SENTARA автоматизує весь цикл: виявлення (8 с) → блокування (5 мс) → класифікація (миттєво) → алерт з досьє оператору (< 1 с). Аналітик витрачає лише 2-5 хвилин на підтвердження алерту замість 2-3 годин на ручний аналіз. При 10 інцидентах на тиждень: економія ~20-30 людино-годин/тиждень, що еквівалентно ~50% робочого часу одного аналітика.

Для кількісної оцінки: при ставці аналітика ~$15/год та 10 інцидентів/тиждень, економія складає 25 год × $15 × 52 тижні = ~$19 500/рік на одному аналітику. З урахуванням вартості SENTARA (€300/рік ≈ $330) - ROI за людино-годинами = $19 500 / $330 = 59×. Тобто навіть без урахування запобігання збиткам від атак, SENTARA окупається 59 разів лише за рахунок автоматизації рутинної роботи аналітиків.

1.5 Постановка задачі

На підставі результатів аналізу ботнет-загроз (підрозділ 1.1), розгляду мультиагентного підходу до кіберзахисту (підрозділ 1.2) та виявлених недоліків наявної системи захисту на підприємстві (підрозділ 1.3) сформульовано задачу роботи: спроєктувати та реалізувати мультиагентну систему виявлення та реагування на ботнет-атаки, що отримала умовну назву SENTARA (Sentinel Tactical Autonomous Response Aegis). Система має містити керуючий вузол та

мережу розподілених автономних агентів, здатних автоматично виявляти, класифікувати та блокувати загрози з урахуванням накопиченої історії взаємодій і забезпечувати безперервне функціонування в умовах часткових відмов інфраструктури.

Функціональні вимоги

ФВ.1. Розподілений збір та аналіз подій безпеки. Агенти системи виконують зчитування записів із системних журналів цільових вузлів (syslog, journald, файл auth.log) та здійснюють їх первинний аналіз на основі набору правил, описаних у декларативному форматі YAML. Локальне виконання аналізу зменшує обсяг даних, що передаються до керуючого вузла, та забезпечує можливість виявлення загроз на рівні окремого вузла.

ФВ.2. Автоматичне блокування джерел загроз. У разі підтвердження факту атаки агент самостійно вносить мережеву адресу її джерела до іменованого набору (set) пакетного фільтра nftables із заздалегідь визначеною тривалістю блокування. Виконання дії блокування не потребує підтвердження оператором, що усуває часовий розрив між виявленням загрози та її нейтралізацією.

ФВ.3. Накопичення історії взаємодій із джерелами загроз. Система зберігає повну послідовність подій, пов'язаних із кожним джерелом, та автоматично класифікує його за одним із чотирьох рівнів ворожості: невідоме джерело, разовий порушник, повторний порушник, постійний порушник. Підвищення рівня класифікації відбувається на основі накопичуваних показників - кількості зафіксованих інцидентів і сумарної оцінки шкідливих дій.

ФВ.4. Функціонування в автономному режимі. За відсутності зв'язку з керуючим вузлом агент продовжує виконання захисних функцій у повному обсязі з використанням локальної бази даних SQLite для збереження накопичених подій та поточного набору правил. Після відновлення з'єднання здійснюється синхронізація накопичених подій із керуючим вузлом за принципом подієво-орієнтованого зберігання (event sourcing), що гарантує повноту та хронологічну впорядкованість переданих даних.ФВ.5. Централізована кореляція подій. Керуючий вузол виконує зіставлення подій, отриманих від усіх агентів, що дозволяє виявляти розподілені атаки, які залишаються непомітними на рівні

окремого вузла.

ФВ.5. Централізована кореляція подій. Керуючий вузол виконує зіставлення подій, що надходять від усіх агентів мережі, з метою виявлення розподілених атак - таких, що залишаються непомітними під час локального аналізу на рівні окремого вузла. Результати кореляції використовуються для оновлення класифікації джерел загроз та формування узагальнених сповіщень оператору.

ФВ.6. Інтерфейс взаємодії з оператором через Telegram. Система забезпечує надсилання сповіщень про виявлені інциденти безпеки та обробку команд оператора у месенджері Telegram. Передбачено такі команди: /status - перегляд загального стану системи, /threats - перелік активних загроз, /block - ручне блокування заданої адреси, /history - перегляд історії взаємодій із джерелами підвищеної ворожості.

Нефункціональні вимоги

Середній час виявлення загрози (MTTD - Mean Time To Detect) не повинен перевищувати 30 секунд, час реакції на блокування - 5 секунд. Система має підтримувати одночасну роботу до 50 агентів. Тривалість автономного функціонування агента без зв'язку з керуючим вузлом - не менше 72 годин. Період зберігання історичних даних - не менше 365 днів.

Цільова архітектура

Запропоновано гібридну архітектуру типу «керуючий вузол - агент». Керуючий вузол реалізовано на основі фреймворку FastAPI з використанням системи керування базами даних PostgreSQL та сховища Redis. До його функцій належать глобальна кореляція подій, ведення бази накопиченої історії загроз та взаємодія з оператором через Telegram-бот. Агенти розгортаються як systemd-сервіси на цільових вузлах і взаємодіють з nftables та локальною базою SQLite, виконуючи збір подій, локальний аналіз та автоматичне блокування. Обмін даними між компонентами відбувається переважно через HTTP-запити, а для передачі подій у реальному часі додатково використовується протокол WebSocket (рисунок 1.3).

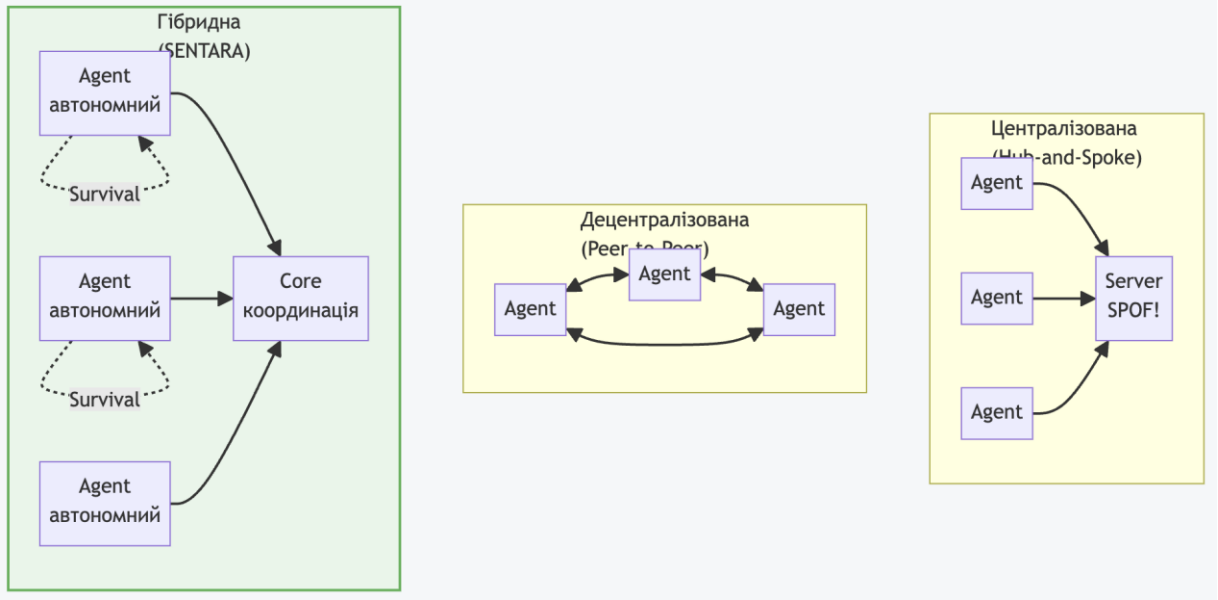


Рисунок 1.3 – Цільова архітектура SENTARA

Поставлена задача вважається вирішеною за умови одночасного досягнення п'яти кількісних показників, що охоплюють якість виявлення, оперативність реагування, відмовостійкість та масштабованість системи.

Перший показник - повнота виявлення (Detection Rate) - характеризує частку успішно виявлених атак серед їх загальної кількості. Цільове значення показника складає не менше 95% для типових категорій атак: підбору паролів служби SSH, сканування мережевих портів, спроб впровадження SQL-коду (SQL Injection) та міжсайтового скриптування (XSS). Одночасно частка хибнопозитивних спрацювань не повинна перевищувати 1% від загальної кількості згенерованих сповіщень.

Другий показник - середній час виявлення загрози (MTTD, Mean Time To Detect) - визначає інтервал між початком шкідливої дії та формуванням відповідного сповіщення оператору. Цільове значення показника - не більше 30 секунд.

Третій показник - час реакції на виявлену загрозу - визначає інтервал між моментом виявлення та внесенням джерела до списку блокування пакетного фільтра nftables. Цільове значення показника - не більше 5 секунд.

Четвертий показник - тривалість автономного функціонування агента в умовах втрати зв'язку з керуючим вузлом - характеризує відмовостійкість

системи. Цільове значення - не менше 72 годин з обов'язковим відновленням повної послідовності накопичених подій після поновлення з'єднання.

П'ятий показник - масштабованість керуючого вузла - визначається кількістю одночасно обслуговуваних агентів за умови збереження прийнятної затримки обробки запитів. Цільове значення - підтримка не менше 10 агентів за затримки відповіді програмного інтерфейсу керуючого вузла на рівні 99-го перцентилю не більше 50 мілісекунд.

Додатково сформульовано три якісних критерії, що характеризують зручність експлуатації та економічну доцільність впровадження системи:

- синтаксис мови опису правил виявлення має бути зрозумілим оператору без попереднього спеціалізованого навчання – час розуміння та ознайомлення для написання першого власного правила не повинен перевищувати 10 хвилин;
- орієнтовна вартість підтримки інфраструктури системи не повинна перевищувати 50 євро на місяць;
- тривалість процедури розгортання одного агента на цільовому вузлі не повинна перевищувати 15 хвилин.

1.6 Висновки до розділу 1

У першому розділі проведено аналіз предметної області та обґрунтовано необхідність розроблення мультиагентної системи протидії ботнет-атакам.

Розглянуто еволюцію ботнет-загроз від ранніх централізованих мереж на основі протоколу IRC до сучасних однорангових ботнетів четвертого покоління, що використовують пристрої інтернету речей та шифровані канали керування. На основі порівняльного аналізу найвідоміших ботнетів і статистичних даних CERT-UA та IBM Security показано, що ботнет-атаки є системною загрозою національного масштабу.

Систематизовано методи виявлення ботнет-активності за чотирма категоріями - сигнатурний, аномальний, мережевий та хостовий аналіз. За

результатами бального оцінювання поширених рішень класу IDS/IPS (Snort, Suricata, Zeek, Wazuh) встановлено, що жодне з них не забезпечує одночасного покриття критично важливих властивостей: автономного функціонування агентів, накопичення історії взаємодій із джерелами, подієво-орієнтованої синхронізації даних та декларативного формату опису правил.На основі виявлених обмежень обґрунтовано доцільність побудови системи за гібридною архітектурою «керуючий вузол - агент».

Обґрунтовано гібридну Core-Agent модель. За результатами аналізу інфраструктури підприємства (Департамент кіберполіції, 15 захищених вузлів) виявлено шість системних недоліків наявної системи захисту, що покладено в основу формулювання вимог.

Сформульовано шість функціональних вимог до системи, нефункціональні вимоги щодо часу виявлення, реагування та автономності, а також п'ять кількісних і три якісних критерії успішного виконання поставленої задачі. Отримані результати становлять основу для проєктування цільової архітектури системи, що розглядається у другому розділі.

2 ПОБУДОВА МУЛЬТИАГЕНТНОЇ СИСТЕМИ ПРОТИДІЇ БОТНЕТ-АТАКАМ SENTARA

2.1 Специфікація та загальна архітектура системи

Проєктування системи SENTARA базується на шести принципах, що безпосередньо адресують недоліки, виявлені у підрозділі 1.3, пріоритеті HTTP-протоколу для обміну даними, подієво-орієнтованому зберіганні даних (Event Sourcing), відокремленні шару формування сповіщень, обов'язковій авторизації агентів оператором, незворотності історії загроз та автономності захисту вузла.

Принцип пріоритетного використання HTTP-протоколу передбачає застосування HTTP як основного каналу обміну даними між агентом та керуючим вузлом завдяки його стійкості до тимчасових мережевих збоїв та відсутності стану з'єднання. Протокол WebSocket використовується додатково для передачі команд, що потребують доставки в режимі реального часу. Після відновлення зв'язку агент здатний передати накопичені події через HTTP POST. Реалізація даного принципу усуває проблему відсутності стійкості до відмов, характерну для існуючих рішень.

Принцип подієво-орієнтованого зберігання даних (Event Sourcing) полягає у збереженні не поточних станів об'єктів, а послідовності подій, що призводять до їх формування. Поточний стан системи розглядається як похідна величина, яка може бути відновлена шляхом відтворення історії подій. Такий підхід забезпечує детерміновану синхронізацію між агентом і керуючим вузлом після відновлення зв'язку та усуває недолік, пов'язаний із відсутністю пам'яті про загрози.

Принцип відокремлення шару формування сповіщень передбачає розділення логіки прийняття рішень щодо реагування на загрози та механізмів формування повідомлень для оператора. Шар сповіщень реалізує адаптивне формулювання повідомлень з урахуванням історії взаємодії та частоти виникнення подій. Це дозволяє зменшити ефект інформаційного перенасичення оператора (alert fatigue) та підвищити ефективність роботи з повідомленнями безпеки.

Принцип обов'язкової авторизації агентів оператором визначає, що

підключення нового агента до системи можливе виключно після його явного підтвердження оператором. Автоматична реєстрація агентів не допускається, що унеможливлює підключення неавторизованих вузлів та знижує ризик компрометації системи через сторонні компоненти.

Принцип незворотності історії загроз встановлює, що записи про взаємодію з джерелами загроз не підлягають видаленню. Рівень ворожості об’єкта може динамічно змінюватися та знижуватися з часом, однак інформація про попередні інциденти зберігається безстроково. Даний принцип доповнює механізм Event Sourcing та усуває недолік, пов’язаний із відсутністю довготривалої пам’яті про загрози.

Принцип автономності захисту вузла передбачає здатність кожного агента виконувати функції виявлення та блокування загроз незалежно від наявності зв’язку з керуючим вузлом. Локальна база даних SQLite та набір правил безпеки забезпечують працездатність механізмів захисту навіть в умовах повної ізоляції. Реалізація цього принципу дозволяє усунути проблему єдиної точки відмови та зменшити реактивний характер захисту системи.

Компоненти системи

Система складається з трьох основних компонентів: керуючого вузла, мережі агентів та інтерфейсу взаємодії з оператором (рисунок 2.1).

Керуючий вузол відповідає за глобальну кореляцію подій, отриманих від агентів, ведення бази історії загроз та забезпечення інтерфейсу взаємодії з оператором. Реалізований на основі фреймворку FastAPI з використанням реляційної СКБД PostgreSQL для довготривалого зберігання даних та сховища Redis для оперативного кешування.

Агенти розгортаються на цільових вузлах і здійснюють збір подій із системних журналів, локальний аналіз за заданим набором правил та автоматичне блокування виявлених джерел загроз через пакетний фільтр nftables. Для збереження локального стану використовується вбудована база даних SQLite, що забезпечує продовження функціонування в умовах відсутності зв'язку з КВ.

Інтерфейс оператора реалізований у вигляді Telegram-бота, інтегрованого з КВ. Забезпечує отримання сповіщень про інциденти, надсилання керуючих

команд та перегляд поточного стану системи.

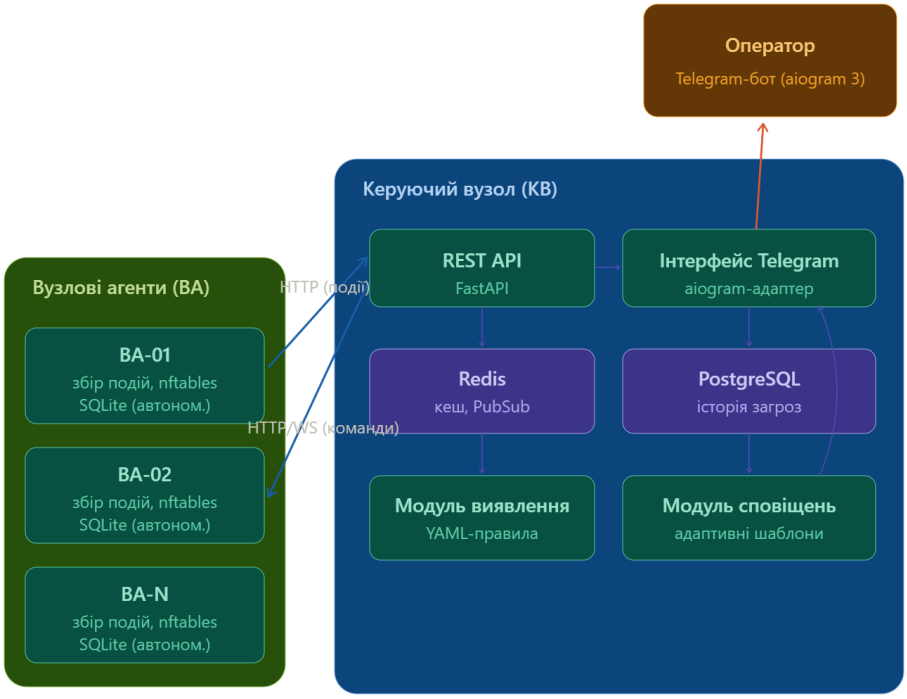


Рисунок 2.1 – Загальна архітектура SENTARA

Вибір технологій визначався критеріями: Python-екосистема, мінімальні зовнішні залежності, підтримка асинхронного I/O (для обробки подій від десятків агентів), та наявність у стандартних репозиторіях Ubuntu/Debian.

Для реалізації системи SENTARA використовується набір сучасних програмних технологій, підібраних відповідно до функціональних вимог архітектури та принципів відмовостійкості.

Керуючий вузол реалізується мовою програмування Python версії 3.11 або новішої, що забезпечує підтримку асинхронного програмування через бібліотеку asyncio, використання анотацій типів та сучасних механізмів розробки. Як основний фреймворк для реалізації REST API використовується FastAPI версії 0.109 або новішої, який забезпечує асинхронну обробку запитів та автоматичне формування документації API. Для взаємодії з базою даних застосовується SQLAlchemy версії 2.0 або новішої з підтримкою асинхронних сесій. Основним сховищем даних є PostgreSQL версії 15 або новішої, у якій зберігаються дані про агентів, події безпеки та історію взаємодії із загрозами. Для кешування

тимчасових даних та організації механізмів обміну повідомленнями використовується Redis версії 7 або новішої. Інтеграція з месенджером Telegram реалізується за допомогою бібліотеки aiogram версії 3.x.

Вузловий агент також реалізується мовою Python версії 3.11 або новішої. Для обміну даними з керуючим вузлом використовується бібліотека aiohttp версії 3.14.0, яка забезпечує асинхронну роботу з HTTP-запитами. Передача команд у режимі реального часу реалізується за допомогою бібліотеки websockets версії 12 або новішої. Для забезпечення автономного режиму роботи використовується локальна база даних SQLite, що входить до стандартного складу Python. Реалізація механізмів блокування мережевих загроз виконується засобами системного мережевого фільтра nftables.

Інфраструктурний рівень системи базується на використанні системного менеджера процесів systemd, який забезпечує запуск, моніторинг та автоматичне відновлення служб. Для ведення журналів подій використовується бібліотека structlog версії 24 або новішої, що підтримує структуроване логування у форматі JSON. Валідація конфігураційних параметрів та вхідних даних виконується за допомогою бібліотеки Pydantic версії 2.0 або новішої.

Механізми виявлення загроз реалізуються на основі декларативної системи правил, описаних за допомогою YAML DSL. Такий підхід дозволяє формалізувати правила виявлення, спростити їх супровід та забезпечити можливість модифікації без внесення змін до програмного коду системи.

Вибір FastAPI обґрунтований: нативна підтримка async/await (критично для обробки подій від десятків агентів одночасно), автоматична генерація OpenAPI документації (спрощує розробку агентів), вбудована валідація через Pydantic (зменшує кількість помилок), та найвища продуктивність серед Python web-фреймворків (до 15 000 req/s на одному ядрі за benchmarks TechEmpower).

Вибір SQLite для агентів (замість PostgreSQL) обґрунтований: нульова конфігурація (файлова БД, не потребує окремого сервісу), мінімальне споживання ресурсів (~500 КБ RAM), вбудованість у Python standard library (модуль sqlite3), та достатня продуктивність для локального кешу подій (до 50 000 INSERT/с).

2.2 Керуючий вузол системи

Керуючий вузол реалізовано за модульним принципом: його функціональність розподілено між дев'ятьма спеціалізованими програмними модулями (engines), кожен з яких відповідає за виконання одного класу задач. Розподіл функцій за окремими модулями забезпечує можливість їх незалежної розробки, тестування та оновлення, а також дозволяє вибірково вмикати або вимикати окремі компоненти залежно від поточних потреб системи. Модулі впроваджуються поетапно - у три черги відповідно до пріоритетності їх функцій. До першої черги віднесено модулі, що забезпечують основну функціональність виявлення та реагування на загрози. Друга черга охоплює модулі, що відповідають за управління життєвим циклом подій та класифікацію джерел загроз. Третя черга включає модулі поглибленого аналізу - кореляційного, статистичного та інтеграції із зовнішніми джерелами даних.

Функціональні можливості системи SENTARA реалізуються поетапно відповідно до пріоритетності компонентів та складності їх впровадження. Такий підхід дозволяє отримати працездатну версію системи на ранніх етапах розробки та поступово розширювати її функціональність.

Перша черга реалізації (MVP) включає модуль виявлення загроз (Detection Engine) та модуль формування сповіщень (Personality Engine). Модуль виявлення забезпечує аналіз подій від агентів за допомогою сигнатурних та порогових правил, описаних у форматі YAML, і формує класифіковані інциденти разом із рекомендованими діями реагування. Модуль формування сповіщень відповідає за адаптивне створення повідомлень оператору з урахуванням історії взаємодії та частоти виникнення подій.

До другої черги належать модуль ескалації (Escalation Engine) та модуль часозалежних переходів (Decay Engine). Модуль ескалації визначає рівень терміновості сповіщень та спосіб їх доставки залежно від критичності інциденту. Модуль часозалежних переходів забезпечує автоматичне завершення строку дії блокувань і поступове зниження рівня ворожості джерел, які протягом тривалого часу не проявляли підозрілої активності.

Третя черга реалізації передбачає впровадження розширених аналітичних та сервісних можливостей системи. До неї належить модуль кореляції (Correlation Engine), призначений для виявлення складних розподілених атак шляхом аналізу подій від декількох агентів. Модуль базових показників (Baseline Engine) формує статистичну модель штатної поведінки вузлів та використовується для виявлення аномалій. Модуль зовнішнього збагачення (Intel Engine) забезпечує інтеграцію із зовнішніми джерелами репутаційних та геолокаційних даних. Модуль звітності (Report Engine) виконує формування аналітичних звітів для оператора, а модуль збереження стану (SEED Engine) створює та розповсюджує зашифровані знімки стану системи для підвищення відмовостійкості та спрощення процедур відновлення.

Обмін даними між керуючим вузлом і агентами реалізовано через REST API за принципом пріоритету HTTP-протоколу. Як основний канал зв'язку використовується REST-інтерфейс на основі HTTP, що забезпечує відсутність стану з'єднання та стійкість до тимчасових мережевих збоїв. Як додатковий канал використовується протокол WebSocket - лише для команд керуючого вузла, що потребують доставки агенту в реальному часі.

По-перше, відсутність стану з'єднання у HTTP дозволяє агенту надсилати накопичені дані відразу після відновлення зв'язку, без необхідності попереднього встановлення сесії.

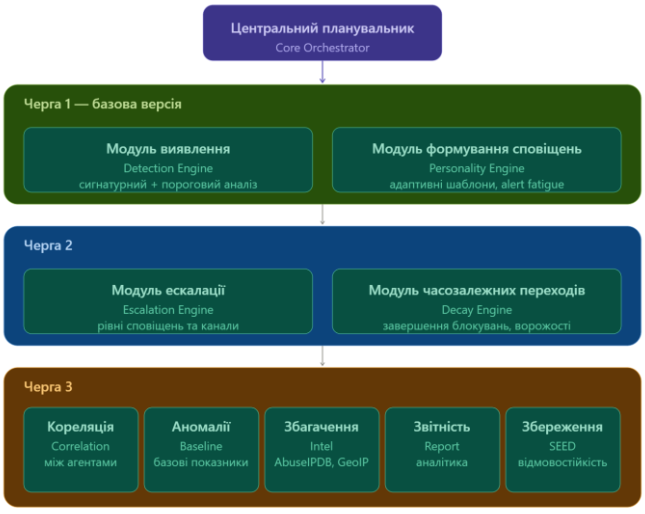


Рисунок 2.2 – Структура програмних модулів керуючого вузла за чергами реалізації

Такий вибір обумовлений двома міркуваннями:

По-друге, наявність WebSocket-каналу не є обов'язковою умовою функціонування системи: у разі його розриву агент продовжує надсилати події керуючому вузлу через стандартні HTTP-запити, а команди можуть бути доставлені під час наступного циклу heartbeat.

Після виходу агента з автономного режиму всі накопичені події передаються до керуючого вузла одним пакетним запитом через окремий груповий ендпоінт інтерфейсу. Це дозволяє виконати синхронізацію за одну транзакцію та зменшити кількість мережевих обмінів.

Перелік основних точок доступу програмного інтерфейсу керуючого вузла та їх призначення наведено у таблиці 2.1.

Таблиця 2.1 – REST API endpoints для керуючого вузла

Метод	Endpoint	Призначення	Auth
POST	/api/v1/events	Приймання подій від агента (груповий формат, до 100 подій за запит)	Agent Token
POST	/api/v1/events/batch	Пакетна синхронізація подій після виходу агента з автономного режиму	Agent Token
GET	/api/v1/rules	Отримання агентом поточного набору правил виявлення	Agent Token
POST	/api/v1/agents/register	Запит на реєстрацію нового агента (потребує підтвердження оператором)	Registration Token
POST	/api/v1/agents/{id}/heartbeat	Періодичний сигнал агента (кожні 30 секунд): поточний стан, метрики, версія правил	Agent Token
GET	/api/v1/threats	Перелік активних загроз для оператора	Operator Token
POST	/api/v1/threats/{id}/block	Ручне блокування адреси за рішенням оператора	Operator Token
GET	/api/v1/status	Загальний стан системи: кількість агентів, активних загроз, стан модулів	Operator Token
GET	/api/v1/nemeses	Перелік джерел загроз, класифікованих як постійні порушники	Operator Token

Автентифікація реалізована за схемою Bearer Token: токен передається у HTTP-заголовку Authorization у форматі Bearer {токен}. У системі використовуються три типи токенів, що відрізняються областю дії та терміном життя.

Реєстраційний токен - одноразовий токен з обмеженим терміном дії, який використовується агентом виключно під час процедури першої реєстрації у керуючому вузлі. Після успішного підтвердження реєстрації оператором цей токен анулюється і не може бути використаний повторно.

Токен агента - постійний токен, що видається керуючим вузлом після підтвердження реєстрації агента оператором. Він прив'язаний до конкретного агента, ідентифікує його у всіх подальших запитах та має обмежений термін дії з можливістю автоматичного оновлення.

Токен оператора - токен, що використовується Telegram-ботом для виконання запитів від імені оператора. Прив'язаний до конкретного ідентифікатора користувача Telegram, що забезпечує однозначну ідентифікацію оператора у системі.

Усі токени генеруються із використанням криптографічно стійкого джерела випадкових чисел; для зберігання у базі даних застосовується хешування алгоритмом SHA-256.Схема бази даних (PostgreSQL)

База даних керуючого вузла зберігає три категорії даних: реєстр агентів, накопичену історію взаємодій з джерелами загроз (записи джерел, зустрічей та активних блокувань) і журнал подій. Ключові таблиці та їх зв'язки зображено на рисунку 2.3.

Таблиця threat_actors зберігає кумулятивну інформацію про кожне джерело загрози: IP-адресу, поточну класифікацію за рівнем ворожості (невідоме джерело, разовий порушник, повторний порушник або постійний порушник), накопичений рівень ворожості (hostility score), кількість виявлених взаємодій, унікальний ідентифікатор джерела (для повторних та постійних порушників), дату першого та останнього виявлення. Таблиця encounters зберігає кожну окрему взаємодію з джерелом загрози: тип події, ідентифікатор правила, що спрацювало, виконану дію, агента, що виявив подію. Таблиця blocks містить активні блокування з

зазначенням адреси, тривалості, причини та агента-виконавця. Таблиця events зберігає повний журнал подій від агентів для ретроспективного аналізу та забезпечення механізму журналювання подій як основного джерела істини (event sourcing).

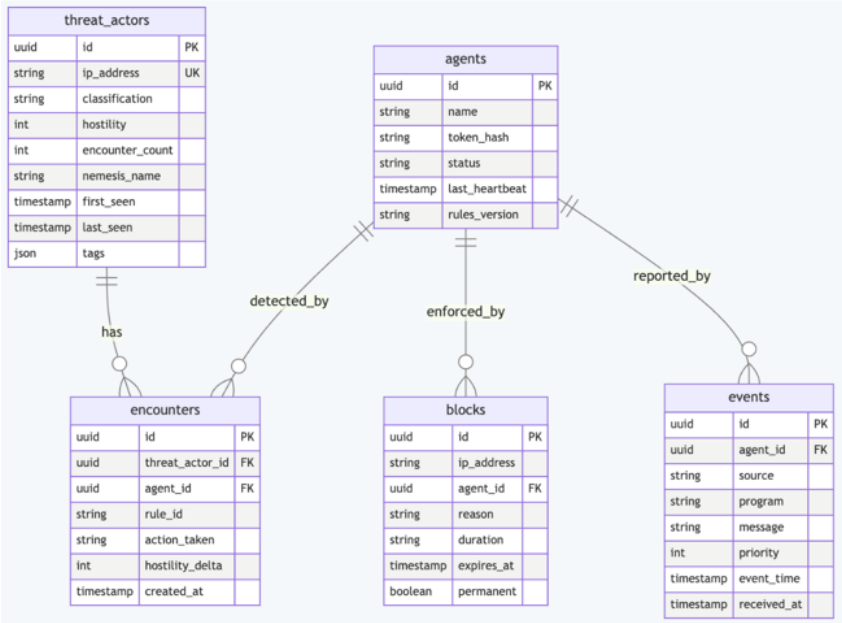


Рисунок 2.3 – ER-діаграма бази даних керуючого вузла

Redis використовується для двох цілей: підтримки лічильників порогів спрацювання правил (наприклад, для правила «5 невдалих спроб SSH-логіну за 60 секунд з однієї адреси» — ключ формату threshold:{rule_id}:{source_ip} з TTL, що дорівнює розміру вікна спостереження) та механізму публікації-підписки для розсилки команд блокування від керуючого вузла до всіх активних агентів у реальному часі через канал sentara:commands.

2.3 Принципи побудови модуля виявлення

Модуль виявлення (Detection Engine) є ключовим компонентом захисту керуючого вузла. Її функціонування побудовано на трьох взаємопов'язаних принципах. Принцип повного перебору правил передбачає, що кожна подія, отримана від агента, послідовно перевіряється на відповідність усім активним

правилам, а не лише до першого збігу. Це забезпечує багаторівневе виявлення, за якого одна подія може активувати декілька правил різного рівня серйозності одночасно - наприклад, спроба несанкціонованого входу може водночас спрацювати правило виявлення підбору пароля та правило сканування портів. Принцип накопичення дій означає, що при одночасному спрацюванні кількох правил усі визначені ними дії виконуються кумулятивно: якщо одне правило передбачає лише журналювання, а інше - блокування адреси, обидві дії будуть виконані. Принцип пріоритету структурного порівняння визначає, що співставлення подій із полями правила (за конкретними ключами та значеннями) має перевагу над регулярними виразами як основний спосіб перевірки, що забезпечує вищу продуктивність і прозорість поведінки системи; регулярні вирази використовуються лише для випадків, коли структурне порівняння є недостатнім (наприклад, для виявлення SQL-ін'єкцій у тілі HTTP-запиту).

Декларативна мова опису правил

Правила виявлення описуються у декларативному форматі YAML як предметно-орієнтована мова (Domain-Specific Language, DSL), що дозволяє створювати та модифікувати правила без зміни програмного коду. Такий підхід розмежовує логіку виявлення та механізм її виконання, забезпечуючи конфігурованість і супроводжуваність системи. Кожне правило містить чотири секції: метадані (ідентифікатор, назва, рівень серйозності, ознака активності), умови спрацювання (співставлення за полями події або регулярним виразом з можливістю виключень), пороги (кількість збігів, часове вікно, поле групування) та дії, що виконуються у разі спрацювання. Узагальнену структуру правила наведено у таблиці 2.2.

Вбудовані правила виявлення

Система поставляється з набором вбудованих правил, що охоплюють найпоширеніші типи атак на серверну інфраструктуру. Набір правил сформовано на основі аналізу звітів CERT-UA щодо актуальних загроз в українському сегменті інтернету та класифікації найкритичніших вразливостей веб-застосунків OWASP Top 10. Перелік вбудованих правил із зазначенням рівня серйозності, порогових значень та визначених дій наведено у таблиці 2.3.

Таблиця 2.2 – Структура YAML Rule DSL

Секція	Поле	Тип	Опис
1	2	3	4
Метадані	id	string	Унікальний ідентифікатор (напр. "ssh_brute_force")
	name	string	Людиночитабельна назва
	severity	int (1-10)	Серйозність: 1 = info, 5 = warning, 8 = high, 10 = critical
	enabled	bool	Увімкнене/вимкнене (підвантаження без перезапуску)
	source	string	Джерело логів: sshd, nginx, kernel, syslog, any
Match	fields	dict	Метчінг за полями: {program: "sshd", message_contains: "Failed password"}
	regex	string	Fallback: регулярний вираз для складних випадків
	exclude	dict	Виключення (whitelist): {source_ip: ["10.0.0.0/8"]}
Threshold	count	int	Кількість збігів для спрацювання (напр. 5)
	window	duration	Часове вікно (напр. "60s", "5m")
	group_by	string	Поле групування (напр. "source_ip")
Actions	log	bool	Записати у журнал подій
	alert	string	Надіслати алерт оператору (severity визначає urgency)
	block	duration	Заблокувати IP через nftables (напр. "24h", "permanent")
	escalate	int	Додати очки ворожості до threat actor
	tag	list	Додати теги до threat actor (напр. ["scanner", "brute_force"])

Таблиця 2.3 – Вбудовані правила виявлення SENTARA

Ідентифікатор	Назва	Рівень	Поріг спрацювання	Дії
1	2	3	4	5
ssh_brute_force	SSH Brute Force Detection	8	5 спроб / 60 с / на 1 IP	alert, block 24h, escalate +20
ssh_success_after_fail	SSH Success After Failures	9	1 успіх після 3+ невдач / 5 хв	alert (urgent), escalate +30

Кінець таблиці 2.3

1	2	3	4	5
port_scan_syn	SYN Port Scan	6	20 портів / 30 с / на 1 IP	alert, block 1h, escalate +10
port_scan_aggressive	Aggressive Port Scan	8	100 портів / 60 с / на 1 IP	alert, block 24h, escalate +25
http_404_flood	HTTP 404 Enumeration	5	50 помилок 404 / 60 с / на 1 IP	log, escalate +5
http_sqli_attempt	SQL Injection Attempt	9	1 збіг (regex)	alert, block 24h, escalate +30, tag: sqli
http_xss_attempt	XSS Attempt	8	1 збіг (regex)	alert, block 12h, escalate +20, tag: xss
dns_tunneling	DNS Tunneling Suspicious	7	100 DNS запитів / 60 с з типом TXT NULL	alert, escalate +15
privilege_escalation	Privilege Escalation	9	1 збіг (виклик sudo від нетипового користувача)	alert (critical), escalate +25
file_integrity	Critical File Modified	8	1 збіг (зміна /etc/passwd, /etc/shadow тощо)	alert (urgent), tag: fim
c2_beaconing	C2 Beaconing Pattern	9	Періодичні з ‘єднання (jitter < 10%) протягом 5 хв	alert (critical), escalate +40, tag: c2

Для ілюстрації формату наведено правило ssh_brute_force з поясненням. Секція метаданих: id = "ssh_brute_force", name = "SSH Brute Force Detection", severity = 8 (high), enabled = true, source = "sshd". Секція match: fields - program: "sshd", message_contains: "Failed password"; exclude - source_ip: ["10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16"] (виключення приватних мереж для уникнення хибних спрацювань від внутрішніх сервісів). Секція threshold: count = 5, window = "60s", group_by = "source_ip" (5 невдалих спроб за 60 секунд з однієї IP). Секція actions: log = true, alert = "SSH brute force from {source_ip}", block = "24h", escalate = 20, tag = ["brute_force", "ssh"].

При кожному записі у файлі auth.log, створеному службою sshd, що містить підстроку «Failed password», підсистема виявлення перевіряє відповідність події умовам секцій match та exclude. Якщо IP-адреса джерела не належить до приватних мереж, відповідний лічильник у Redis (ключ формату threshold:ssh_brute_force:{source_ip}, TTL = 60 с) атомарно інкрементується. При досягненні значення лічильника 5 виконується послідовність визначених у правилі дій: запис у журнал подій, надсилання алерту оператору через Telegram-бот (модуль формування сповіщень обирає відповідний шаблон повідомлення з підстановкою контекстних змінних), блокування адреси на 24 години засобами nftables, додавання 20 одиниць до накопиченого рівня ворожості відповідного запису про джерело загрози (або створення нового запису про невідоме джерело, якщо адреса фіксується вперше), додавання тегів brute_force та ssh. За аналогічним механізмом працюють і решта вбудованих правил. Оператор має можливість створювати власні правила у директорії rules/custom/; такі правила автоматично завантажуються при наступному гарячому перезавантаженні без перезапуску системи.

Механізм відліку порогів реалізовано на основі атомарних операцій Redis з використанням командного примітиву INCR у поєднанні з параметром TTL (Time To Live, час життя запису). Для кожної унікальної комбінації ідентифікатора правила та значення поля групування формується окремий ключ у Redis з автоматичним терміном життя, що дорівнює розміру часового вікна правила. Команда INCR атомарно збільшує значення лічильника та повертає оновлене

значення; якщо ключ не існував до моменту виклику, Redis створює його зі значенням 1 і одночасно встановлює визначений TTL. У разі досягнення лічильником порогового значення, заданого у правилі, ініціюється виконання набору визначених дій. По завершенні часового вікна Redis автоматично видаляє ключ, скидаючи лічильник у початковий стан.

Такий підхід забезпечує одночасно кілька суттєвих властивостей: атомарність операції (інкремент і повернення значення виконуються як єдина команда Redis, що виключає стани гонитви даних при паралельному оновленні з різних потоків); автоматичне скидання лічильника без необхідності періодичного очищення з боку прикладного коду; мінімальну затримку обробки (Redis працює з даними в оперативній пам'яті, тривалість однієї операції — близько 0,1 мс); та високу пропускну здатність (типове сучасне обладнання дозволяє виконувати понад 100 000 операцій інкременту за секунду на одному ядрі процесора). Для розгортання з 50 агентами, кожен з яких генерує до 100 подій за секунду, при 11 активних правилах розрахункове навантаження складає близько 55 000 операцій інкременту за секунду, що знаходиться у межах продуктивності одного екземпляру Redis і не потребує горизонтального масштабування підсистеми зберігання станів.

При надходженні події від агента до керуючого вузла підсистема виявлення виконує наступну послідовність кроків: (1) розбір події та перетворення її у внутрішнє структуроване представлення; (2) послідовний перебір усіх активних правил; (3) для кожного правила — перевірка відповідності події умовам секції співставлення (структурне порівняння за полями події або порівняння за регулярним виразом); (4) у разі збігу — перевірка порогу спрацювання шляхом інкременту відповідного лічильника у Redis з часом життя, що дорівнює розміру вікна спостереження; (5) у разі досягнення порогового значення — виконання визначеного у правилі набору дій (журналювання, надсилання алерту, блокування адреси, оновлення рівня ворожості, тегування);

Процес обробки подій представлений на рисунку 2.4.

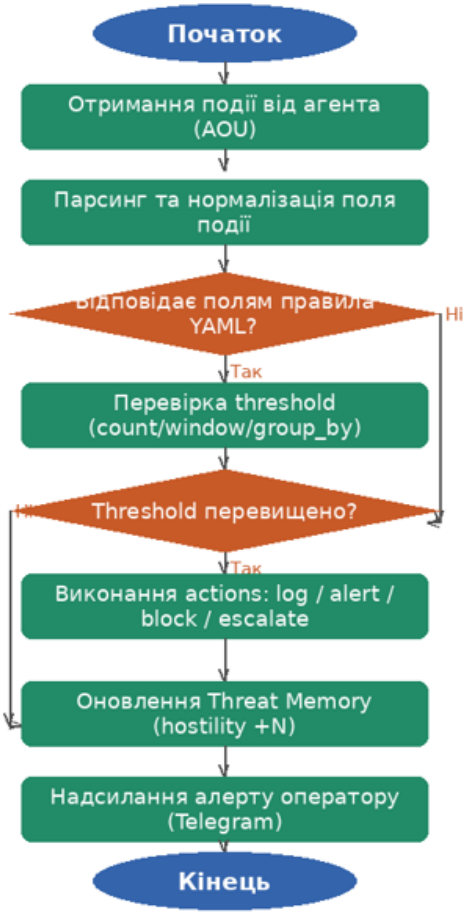


Рисунок 2.4 – Алгоритм роботи модуля виявлення

(6) модуль формування сповіщень обирає відповідний шаблон повідомлення для алерту з підстановкою контекстних змінних. Уся послідовність виконується асинхронно і займає менше 10 мс на одну подію у типовому режимі навантаження.

2.4 Класифікація джерел загроз та механізм накопиченої пам’яті

Однією з ключових відмінностей системи SENTARA від класичних систем виявлення вторгнень є наявність механізму накопиченої пам'яті взаємодій з джерелами загроз. Традиційні рішення, такі як Snort та Suricata, аналізують кожну подію ізольовано: при збігу події з сигнатурою генерується алерт, в іншому випадку подія ігнорується, при цьому інформація про попередні взаємодії з тим самим джерелом не береться до уваги. У запропонованій системі для кожної IP-

адреси, від якої зафіксовано хоча б одну подію безпеки, зберігається повна історія взаємодій, на основі якої виконується класифікація джерела та приймаються рішення щодо подальших дій. Повторна шкідлива активність з тієї ж адреси розглядається не як ізольований інцидент, а як продовження взаємодії з уже відомим системі джерелом загрози.

Кожна IP-адреса, що з'являється у подіях безпеки, отримує запис у таблиці threat_actors з кумулятивними полями: hostility (числовий рівень ворожості, зростає при кожній шкідливій дії), encounter_count (загальна кількість зустрічей), classification (поточна класифікація), first_seen та last_seen (часові межі), tags (набір тегів: scanner, brute_force, sqli, c2 тощо).

Система класифікації загроз

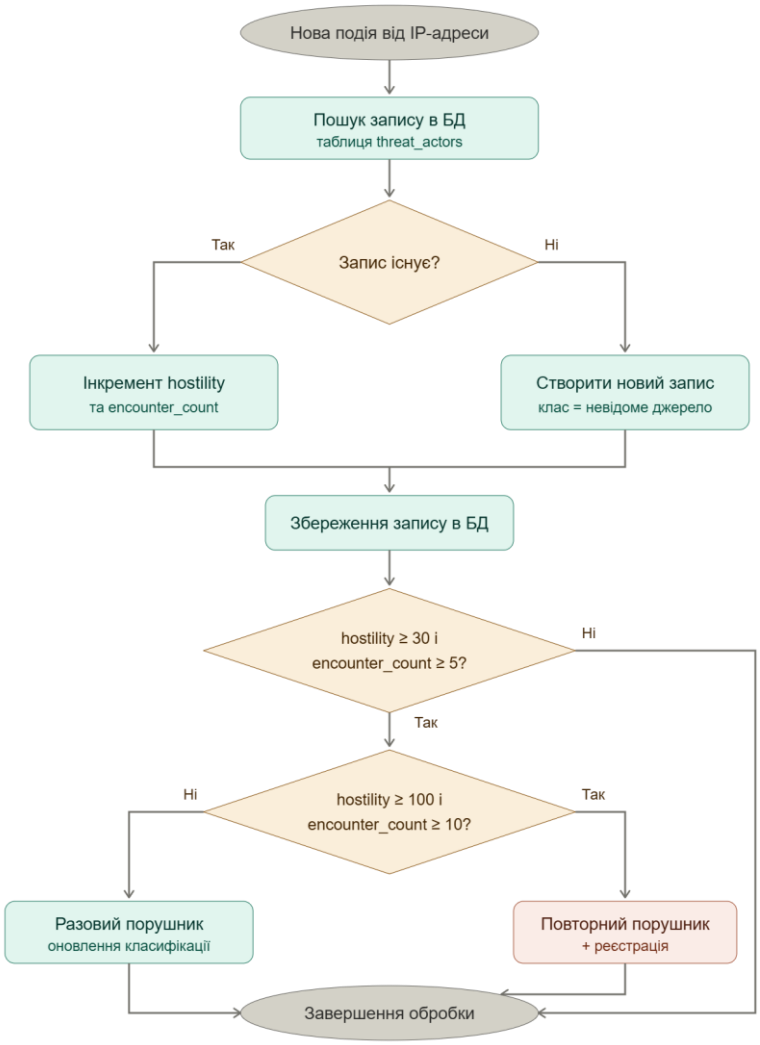


Рисунок 2.5 – Алгоритм класифікації джерел загроз

Система SENTARA класифікує кожне джерело загрози за чотирирівневою ієрархією на основі двох метрик: накопиченого рівня ворожості та загальної кількості зафіксованих взаємодій. Рівні класифікації, порогові значення метрик для переходу між рівнями та визначена поведінка системи для кожного рівня наведені у таблиці 2.4.

Таблиця 2.4 - Ієрархія класифікації загроз

Клас джерела загрози	Рівень загрози	Кількість взаємодій	Поведінка системи	Типовий приклад
Невідоме джерело	0-29	1-4	Стандартне спостереження за загальними правилами. Більшість легітимних адрес залишаються у цьому класі	Перший невдалий SSH-логін з раніше невідомої адреси
Разовий порушник	30-99	5-9	Підвищена увага, знижені порогові значення спрацювання правил. Інформація про джерело передається активним агентам	Адреса, з якої зафіксовано 5+ невдалих SSH-спроб у поєднанні зі скануванням портів
Повторний порушник	100+	10+	Джерелу присвоюється унікальний внутрішній ідентифікатор. Оператор отримує персональні алерти. Усі взаємодії логуються окремим потоком	Сканер, що періодично повертається до інфраструктури протягом тижнів
Постійний порушник	100+	10+ протягом 90+ днів	Джерело загрози з тривалим періодом активності (понад 90 днів). Запис не видаляється з історії за жодних умов	APT-актор, що проявляє активність протягом місяців

Переходи між класами відбуваються автоматично на основі поточних значень накопичених метрик. При досягненні джерелом значень, що відповідають класу повторного порушника (рівень ворожості не менше 100, кількість зафіксованих взаємодій не менше 10), система ініціює процедуру реєстрації

стійкого джерела загрози: присвоює джерелу унікальний внутрішній ідентифікатор у форматі мнемонічного коду (для зручності оператора при подальших згадках у логах і алертах), формує консолідоване досьє джерела (загальна кількість взаємодій, типи зафіксованих атак, хронологія активності, поточний стан блокування) та надсилає оператору відповідне сповіщення через Telegram-бот.

Механізм оцінки рівня ворожості

Рівень ворожості (hostility) - це накопичувальний числовий показник, що відображає інтегральну оцінку системою ступеня небезпеки конкретної IP-адреси. Значення показника зростає при кожній зафіксованій шкідливій дії на величину, визначену у полі escalate відповідного правила виявлення, та повільно зменшується з часом за відсутності нових інцидентів (механізм часового послаблення, decay). Така схема реалізує принцип поступового зниження активного рівня загрози при повному збереженні історії взаємодій: за відсутності нової активності оцінка джерела поступово повертається до нейтральних значень, проте кожна попередня взаємодія залишається у журналі назавжди.

Параметри механізму часового послаблення: значення рівня ворожості зменшується на 1 одиницю за кожні 24 години відсутності зафіксованої активності з боку джерела. Нижня межа показника - 0 (значення не може набувати від'ємних значень). Для адрес, що перебувають у стані блокування, механізм послаблення тимчасово припиняє роботу: блокування «фіксує» поточне значення рівня ворожості до моменту зняття блокування. Будь-яка нова шкідлива дія скидає таймер послаблення. Як наслідок, повторний порушник, який припинив активність на 100 днів, втратить 100 одиниць рівня ворожості; проте якщо його накопичене значення складало, наприклад, 200, він збереже статус повторного порушника (поріг - 100). Повне зниження рівня ворожості від статусу повторного порушника до класу разового порушника потребує понад 100 днів повної відсутності активності.

Оновлення рівня ворожості реалізовано на основі атомарної операції PostgreSQL з використанням оператора UPDATE ... RETURNING: UPDATE threat_actors SET hostility = hostility + delta WHERE ip_address = ? RETURNING

hostility, classification. Атомарність операції гарантує коректність результату навіть за умови паралельного оновлення з кількох потоків обробки подій. Після кожного оновлення виконується перевірка необхідності зміни класифікаційного рівня джерела. Якщо нове значення рівня ворожості перетнуло один із порогових рівнів (30 або 100), а значення кількості взаємодій відповідає визначеним умовам, класифікація оновлюється. При переході джерела до класу повторного порушника генерується внутрішня подія реєстрації стійкого джерела загрози, що активує модуль формування сповіщень для створення внутрішнього ідентифікатора та консолідованого досьє джерела.

Механізм часового послаблення реалізовано як окремий фоновий процес, що працює на основі асинхронного циклу подій бібліотеки asyncio та виконується щоденно о 00:00 за UTC. Процес виконує SQL-запит виду UPDATE threat_actors SET hostility = GREATEST(hostility - 1, 0) WHERE last_seen < NOW() - INTERVAL '24 hours' AND NOT EXISTS (SELECT 1 FROM blocks WHERE blocks.ip_address = threat_actors.ip_address AND expires_at > NOW()), який зменшує рівень ворожості на одиницю для всіх записів, що не мають активних блокувань і за якими не зафіксовано взаємодій за останні 24 години. Використання функції GREATEST(..., 0) гарантує, що значення показника не опуститься нижче нуля. Такий підхід забезпечує автоматичне поступове зниження оцінки джерел, які припинили шкідливу активність, без необхідності ручного втручання оператора.

При спрацюванні правила з визначеною дією блокування агент додає IP-адресу до іменованого набору (set) фаєрволу nftables з обмеженим часом життя запису. Типові значення тривалості блокування у вбудованих правилах: 1 година для виявленого сканування портів, 24 години для атак типу підбору пароля та SQL-ін'єкцій, постійне блокування - лише за прямим рішенням оператора.

Після закінчення визначеного періоду блокування знімається автоматично завдяки вбудованому механізму обмеження часу життя записів у наборах nftables.

Життєвий цикл порушника представлений на рисунку 2.5

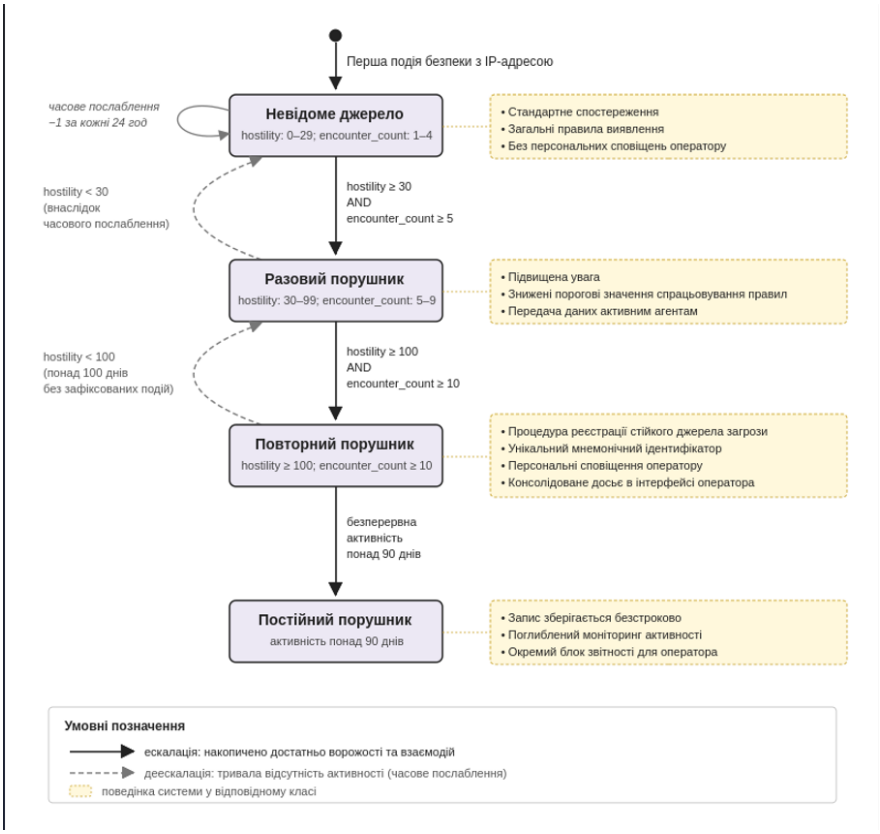


Рисунок 2.6 – Життєвий цикл порушника

Оператор має можливість вручну зняти активне блокування або встановити нове за допомогою команд /unblock та /block у Telegram-боті.

2.5 ВА: автономний захист вузлів

Вузловий агент є легким Python-сервісом, що розгортається на захищеному вузлі як systemd-юніт. Він складається з чотирьох внутрішніх модулів: модуль збору подій (читає системні журнали), модуль локального аналізу (перевіряє події за кешем правил), модуль реагування (виконує блокування джерел через nftables) та модуль зв'язку (HTTP/WebSocket-клієнт для обміну з керуючим вузлом).

Агент споживає мінімальні ресурси: близько 30 МБ оперативної пам'яті у стані очікування, близько 50 МБ під час обробки подій, менше 1% завантаження ЦП за нормального навантаження (до 100 подій за секунду). Це дає змогу розгортати агентів навіть на VPS з обсягом пам'яті 512 МБ без відчутного впливу

на основні сервіси захищеного вузла.

Збір подій із системних журналів

Модуль збору відстежує зміни у файлах журналів за допомогою бібліотеки watchfiles, побудованої на механізмі inotify ядра Linux, та перетворює нові рядки у структурований формат - словник з полями timestamp, source, program, message, priority. Підтримувані джерела: /var/log/syslog (загальні системні повідомлення), /var/log/auth.log (події SSH, sudo, PAM), журнал systemd-journald, а також довільні шляхи, що задаються у конфігурації (наприклад, /var/log/nginx/access.log для веб-серверів). Синтаксичний розбір виконується набором регулярних виразів для кожного типу журналів (формати syslog RFC 5424, auth.log, nginx combined).

Локальний аналіз та блокування

Модуль локального аналізу перевіряє кожну подію за підмножиною правил декларативної мови YAML, отриманих від керуючого вузла через ендпоінт GET /api/v1/rules. Правила кешуються локально та оновлюються при кожному циклі сигналу життєздатності - раз на 30 секунд. У разі збігу події з правилом, що містить дію block, модуль реагування негайно блокує джерело через nftables: додає IP-адресу до іменованого набору sentara_blocklist із заданим часом дії (наприклад, timeout 24h). Час виконання блокування становить менше 5 мілісекунд - це принципово швидше за очікування рішення від керуючого вузла мережею.

Паралельно модуль зв'язку надсилає подію до керуючого вузла через POST /api/v1/events для глобальної кореляції та оновлення накопиченої пам'яті взаємодій. У такий спосіб блокування відбувається локально на рівні мілісекунд, а глобальна аналітика - централізовано на рівні секунд.

Автономний режим

Автономний режим вмикається автоматично при втраті зв'язку з керуючим вузлом - після трьох послідовних невдалих сигналів життєздатності (90 секунд). У цьому режимі агент: продовжує збирати та аналізувати події за локальним кешем правил; виконує блокування самостійно; зберігає всі події у локальну базу SQLite (таблиця survival_events); повторює спробу підключення до керуючого вузла кожні 60 секунд.

Після відновлення зв'язку агент виконує синхронізацію - пересилає всі накопичені події через POST /api/v1/events/batch. Керуючий вузол обробляє події хронологічно, оновлює накопичену пам'ять взаємодій та виконує кореляцію з подіями від інших агентів за той самий період. Після успішної синхронізації локальна база SQLite очищується.

Ключовим архітектурним рішенням для автономного режиму є подієво-орієнтована синхронізація. Замість того щоб синхронізувати стан системи (що вимагало б складного узгодження конфліктів при одночасних змінах на агенті та керуючому вузлі), синхронізуються самі події - факти, що відбулися. Керуючий вузол послідовно обробляє кожну подію в хронологічному порядку, викликаючи модуль виявлення. Такий підхід забезпечує: (1) детермінованість - однакова послідовність подій завжди дає однаковий стан; (2) ідемпотентність - повторне надсилання тих самих подій не змінює результату завдяки дедуплікації за event_id; (3) повний журнал аудиту - жодна подія не втрачається, навіть якщо вона відбулася під час автономної роботи.

Формат пакетного запиту: POST /api/v1/events/batch містить JSON-масив подій (до 1000 за один запит). Кожна подія включає event_id (UUID, що генерується агентом), agent_id, timestamp (час самої події, а не час її надсилання), source, program, message, priority та необов'язкові поля parsed_fields і matched_rules - якщо агент уже виконав локальний аналіз. Керуючий вузол повертає відповідь зі статусом 207 Multi-Status: для кожної події - accepted або duplicate (якщо її вже було оброблено раніше). За мережевого збою під час пакетної синхронізації агент повторно надсилає лише ті події, які не було підтверджено.

Гранична вимога до автономного режиму: збереження повної працездатності не менше 72 годин без зв'язку з керуючим вузлом. Розмір локальної бази SQLite обмежено 100 МБ; за середнього розміру події близько 200 байтів це відповідає приблизно 500 000 подій, що достатньо для понад 72 годин роботи за швидкості 100 подій за секунду. Дисковий простір контролюється ротацією за принципом «перший прийшов - перший вийшов»: при досягненні 90% ліміту найстаріші події видаляються. Параметри автономного режиму зведено в таблиці 2.5.

Таблиця 2.5 – Параметри агента у автономному режимі

Параметр	Значення	Обґрунтування
1	2	3
Тригер активації	3 невдалих сигнали життєздатності (90 с)	Запобігає хибному спрацьовуванню за короткочасних мережевих збоїв
Інтервал повторного підключення	60 с	Баланс між швидкістю відновлення та навантаженням на мережу
Обмеження локальної бази SQLite	100 МБ	~500 000 подій, достатньо для понад 72 годин
Ротація подій	При 90% ліміту	Гарантує дисковий простір для нових подій
Джерело правил	Локальний кеш YAML	Останні правила від керуючого вузла, оновлюються при відновленні зв'язку
Блокування	Автономне (nftables)	Захист вузла не залежить від зв'язку з керуючим вузлом

На рисунку 2.6 представлено життєвий цикл вузлового агента - три послідовні стани (нормальний режим, автономний режим, синхронізація після відновлення зв'язку), якими агент керує самостійно під час роботи на захищеному вузлі (сервері, робочій станції, маршрутизаторі тощо).

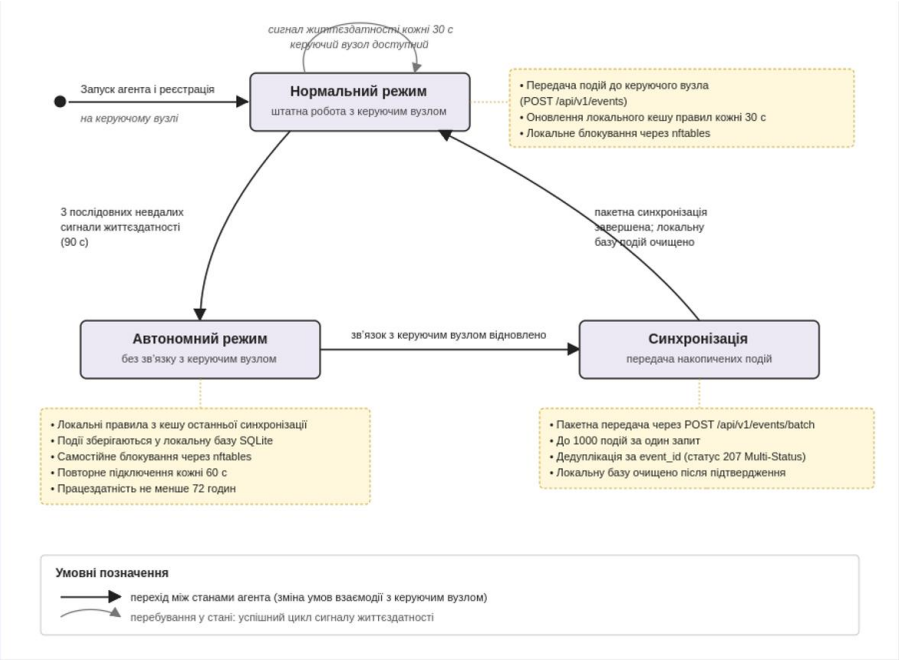


Рисунок 2.7 - Життєвий цикл вузлового агент

2.6 SEED: система збереження стану

Концепція SEED (Secure Encrypted Essential Data) - це зашифрований знімок критичного стану системи SENTARA, що уможливлює повне відновлення системи після втрати керуючого вузла. SEED містить такі складники: накопичену пам'ять взаємодій (усі записи з таблиці threat_actors - джерела загроз, історію зустрічей, рівні ворожості, ідентифікатори стійких джерел), стан зв'язку з оператором (ідентифікатор оператора, поточний рівень зв'язку), реєстр зареєстрованих агентів зі службовими токенами, стан модуля формування сповіщень (поточний режим звернень та накопичену історію взаємодій з оператором) та конфігурацію системи.

До SEED не входять: сирі журнали подій (через значний обсяг; за потреби відновлюються з агентів), тимчасові кешовані дані Redis та правила виявлення (зберігаються у Git-репозиторії вихідного коду системи). Завдяки цьому SEED залишається компактним: від 150 КБ до 3 МБ у стиснутому зашифрованому вигляді.

Шифрування та розповсюдження

SEED шифрується за алгоритмом AES-256-GCM (Galois/Counter Mode), що одночасно забезпечує конфіденційність і автентичність даних. Ключ шифрування формується з біометричного підтвердження оператора за функцією деривації ключів PBKDF2 (Password-Based Key Derivation Function 2, RFC 8018) з 600 000 ітерацій SHA-256 та випадковою сіллю довжиною 32 байти. Кількість ітерацій обрано відповідно до чинних рекомендацій OWASP: не менше 600 000 для PBKDF2-SHA256, що забезпечує стійкість до атак повного перебору навіть на сучасних графічних процесорах (близько 10 000 спроб за секунду на RTX 4090)

Зашифрований SEED розповсюджується на всі зареєстровані агенти: кожен агент зберігає повну копію у локальній базі SQLite (таблиця seed_snapshots). Оновлення SEED відбувається за двома сценаріями: за розкладом - щодня о 02:00 за UTC - та після кожної значущої зміни стану системи (поява нового стійкого джерела загрози, зміна рівня зв'язку з оператором, реєстрація або вилучення агента). Передача оновленого знімка відбувається з ініціативи агента: під час

чергового сигналу життєздатності агент порівнює значення заголовка X-SEED-Version із локальною версією; якщо версії розбіжні, агент запитує новий знімок через GET /api/v1/seed.

Розмір SEED залежить від обсягу накопиченої пам'яті взаємодій: від 150 КБ для нової системи (менше 100 записів у таблиці джерел загроз) до 3 МБ для зрілої системи (понад 10 000 записів, понад 100 стійких джерел). Попереднє стиснення алгоритмом zlib з рівнем 6 зменшує обсяг даних у 3–5 разів до моменту шифрування. Для типового стиснутого знімка обсягом 2 МБ передача одному агенту через HTTP займає менше однієї секунди навіть на каналі з пропускною здатністю 10 Мбіт/с.

Процедура відновлення

У разі повної втрати керуючого вузла процедура відновлення системи виконується у п'ять кроків: (1) оператор розгортає новий керуючий вузол з Git-репозиторію (вихідний код та поточний набір правил виявлення); (2) оператор виконує автентифікацію за біометричним підтвердженням; (3) новий керуючий вузол запитує SEED від будь-якого доступного агента через HTTP; (4) отриманий знімок розшифровується ключем, що деривується з біометричного підтвердження оператора; (5) накопичена пам'ять взаємодій, стан зв'язку з оператором та реєстр агентів відновлюються - система продовжує роботу з усіма раніше накопиченими знаннями. Необхідними умовами процедури відновлення є наявність Git-репозиторію вихідного коду, хоча б одного працездатного агента та фізичної присутності оператора для проходження біометричного підтвердження.

2.7 Інтерфейс оператора: Telegram-бот

Архітектура інтерфейсу оператора

Інтерфейс оператора реалізовано як Telegram-бот на основі фреймворку aiogram 3.x (асинхронна архітектура, скінченні автомати станів, конвеєр проміжного програмного забезпечення). Вибір Telegram як платформи зумовлений такими чинниками: присутність клієнтського застосунку на

мобільних пристроях (оператор отримує сповіщення цілодобово), підтримка форматування повідомлень (Markdown, вбудовані клавіатури з кнопками дій), вбудована система push-сповіщень, наскрізне шифрування каналу за протоколом MTProto, а також відсутність потреби розробляти окремий мобільний застосунок.

Бот підтримує два типи взаємодії: вхідні сповіщення в напрямку «система -> оператор» та команди в напрямку «оператор -> система». Автентифікація оператора прив'язана до унікального ідентифікатора Telegram User ID - взаємодіяти з ботом може лише один заздалегідь авторизований користувач.

Набір команд спроєктовано за принципом мінімальної достатності: оператор повинен мати змогу отримати повну картину стану системи та виконати критичні дії без доступу до командного рядка сервера. Перелік підтримуваних команд наведено в таблиці 2.6.

Таблиця 2.6 – Команди Telegram-бота SENTARA

Команда	Опис	Приклад відповіді
/status	Загальний стан системи: кількість агентів (підключені / недоступні), активні джерела загроз, поточний стан настрою, час безперервної роботи	«Стан системи: 5 агентів підключено, 3 активних джерела загроз, mood: Vigilant, час безперервної роботи: 14 дн 6 год»
/threats	Перелік останніх 10 джерел загроз із класифікацією, рівнем ворожості та останньою зафіксованою дією	«1. 185.220.x.x [разовий порушник] hostility: 67, остання дія: підбір SSH-паролю, 2 год тому»
/nemeses	Перелік стійких джерел загроз (повторні та постійні порушники) із детальним досьє	"<IDENTIFIER> - hostility: 234, зустрічей: 47, активний 23 дні, tags: scanner, c2"
/block <ip>	Ручне блокування IP-адреси на всіх агентах (безстрокове до виконання /unblock)	«Заблоковано 185.220.x.x на 5 агентах. Тривалість: безстрокова»
/unblock <ip>	Зняття активного блокування	«Знято блокування 185.220.x.x на 5 агентах»
/agents	Перелік агентів: ім'я, стан, час останнього сигналу життєздатності, версія набору правил	«BA-01: підключений, сигнал життєздатності 12 с тому, версія правил 2.3»
/approve <id>	Підтвердження реєстрації нового агента	«BA-04 підтверджено та зареєстровано»

Вхідні сповіщення оператору форматуються з урахуванням рівня серйозності події та її контексту. Кожне сповіщення містить: рівень серйозності (інформаційне, попереджувальне, критичне; кодується відповідною піктограмою у тексті повідомлення), назву правила, що спрацювало, IP-адресу джерела загрози та його поточний клас, виконану дію (заблоковано, зареєстровано у журналі, передано на розгляд оператора) і покликання на повне досьє джерела. Для сповіщень про повторні та постійні порушники додається персональне досьє: мнемонічний ідентифікатор джерела, загальна кількість зафіксованих взаємодій, перелік тегів, хронологія останніх дій.

Модуль формування сповіщень адаптує тон повідомлення залежно від поточного стану настрою (mood) системи та рівня зв'язку з оператором. У стані Vigilant (базовий, за стандартного рівня зв'язку): «Виявлено загрозу. Підбір SSH-паролю з 185.220.x.x. Заблоковано на 24 години». У стані Warm (за високого рівня зв'язку з оператором): «Біля периметра системи зафіксовано активність. Підбір SSH-паролю - атаку зупинено. 185.220.x.x не повернеться найближчі 24 години». Зміна тону не впливає на функції безпеки - рішення про блокування ухвалюється модулем виявлення незалежно від модуля формування сповіщень.

2.8 Модуль адаптивного формування сповіщень

Контекст: проблема перенасичення сповіщеннями

Однією з найбільш задокументованих проблем експлуатації систем виявлення вторгнень у центрах операційного моніторингу безпеки (Security Operations Center, SOC) є явище інформаційного перенасичення оператора, що в спеціалізованій літературі отримало назву alert fatigue — втома від сповіщень [27; 41]. За даними галузевих звітів, типовий оператор SOC отримує від кількох сотень до кількох тисяч сповіщень безпеки щодня, при цьому 30–50 відсотків з них залишаються без розгляду через когнітивне виснаження [42]. Однотипне форматування сповіщень, інваріантне щодо реального рівня загрози, з часом призводить до ефекту звикання: оператор перестає розрізняти критичні та

малозначущі події, що збільшує ризик пропуску реального інциденту та подовжує час реагування.

Запропоноване в роботі рішення цієї проблеми - модуль адаптивного формування сповіщень (у вихідному коді системи має технічний ідентифікатор Personality Engine). На відміну від традиційних підходів, що зосереджуються виключно на пріоритизації та дедуплікації сповіщень, цей модуль додатково варіює спосіб подання інформації: тон повідомлень, ступінь деталізації, формулювання, контекстні маркери. Завдяки цьому досягається ефект подолання сприйнятної одноманітності - повідомлення про критичну подію відрізняється від рутинного не лише позначкою серйозності, а й структурою тексту, що додатково привертає увагу оператора. Модуль не впливає на функції виявлення та реагування - рішення щодо блокування ухвалюються модулем виявлення незалежно - і виконує виключно роль адаптивного шару подання даних.

Модуль адаптивного формування сповіщень оперує двома накопичувальними змінними стану: поточним станом настрою системи (mood) та рівнем зв'язку з оператором (bond level). Стан настрою змінюється динамічно у відповідь на події у системі: успішне блокування загрози, тривала відсутність зафіксованих інцидентів, масовий вхідний потік подій, втрата зв’язку з агентом тощо. Рівень зв'язку зростає поступово завдяки регулярній взаємодії оператора з системою та виконанню періодичних службових процедур (щоденне підтвердження повноважень оператора, контроль системи у періоди низької активності).

Комбінація поточних значень обох змінних визначає, який саме шаблон повідомлення обирається з бази шаблонів для формування сповіщення. У такий спосіб система забезпечує контекстне різноманіття вихідних повідомлень без втручання людини у процес їх редагування.

Стани настрою (Mood States)

Модуль підтримує вісім станів настрою, згрупованих у чотири категорії за характером подій, що зумовлюють їх активацію. Перелік станів, умови переходу між ними та визначений характер комунікації наведено в таблиці 2.7.

Таблиця 2.7 – Стани настрою (Mood States) SENTARA

Категорія	Стан	Умова арктивації	Характер комунікації
Базовий	Vigilant	За замовчуванням; повернення після інших станів	Контрольований, уважний, лаконічний
Позитивний	Satisfied	Успішне блокування загрози, нейтралізація стійкого джерела	Стриманий, упевнений
	Calm	Тривала відсутність зафіксованих подій (понад 4 години)	Спокійний, задумливий
	Warm	Інтенсивна позитивна взаємодія з оператором (доступний за bond ≥ Trusted)	Відкритий, менш формальний
Активний	Alert	Виявлено нову загрозу	Зосереджений, короткий, прямий
	Focused	Загроза рівня повторного або постійного порушника	Персональний, інтенсивний
	Engaged	Масовий вхідний потік подій (понад 10 подій за хвилину з різних IP-адрес)	Концентрований, рішучий
Негативний	Somber	Втрата зв’язку з агентом, невдале відновлення, тривала ізоляція	Притишений, лаконічний

Рівень зв'язку з оператором (Bond Level) - це накопичувальний числовий показник, що відображає інтенсивність взаємодії оператора з системою за всю історію їх співпраці. Показник зростає за такі дії: щоденна перевірка стану системи (+5 одиниць), підтвердження сповіщень (+10), виконання періодичних службових процедур (+25–50), безперервна тривала експлуатація системи (+1 за добу).

Поточне значення рівня зв'язку впливає на роботу модуля у трьох аспектах. По-перше, визначає, які саме шаблони повідомлень доступні (зокрема, шаблони з тону Warm активуються лише за рівня зв'язку не нижче Trusted). По-друге, регулює глибину деталізації сповіщень: за низького рівня модуль формує повніший контекст для нового оператора, за високого - лаконічніші повідомлення для досвідченого. По-третє, керує доступністю окремих категорій періодичних службових процедур.

Шкала рівня зв'язку містить п'ять градацій: Silence (0 одиниць - початковий стан), Familiar (від 100), Trusted (від 300), Devoted (від 600), Mature (від 1000 одиниць). Запропонована шкала ґрунтується на принципі поступового нарощування довіри: новий оператор отримує детальніші, явно структуровані повідомлення, що зменшує ризик неправильної інтерпретації, а досвідчений оператор - стиснені повідомлення, що зменшують когнітивне навантаження за тривалої експлуатації системи.

Базу шаблонів повідомлень модуля становлять понад 200 текстових шаблонів, розподілених за вісьмома категоріями за тематикою сповіщення: взаємодія з оператором (привітання, прощання, підтвердження команд), виявлення та реагування на загрози (фіксація події, блокування, ескалація), робота зі стійкими джерелами загроз (реєстрація, оновлення досьє), звіти про стан системи (агреговані метрики), контекстні шаблони відповідно до поточного стану настрою, повідомлення режиму автономної роботи, періодичні службові процедури, рефлексивні службові повідомлення (аналітичні зведення за період). Усі шаблони підтримують механізм підстановки змінних: на місце спеціальних маркерів {threat_ip}, {threat_name}, {agent_name}, {encounter_count} та інших підставляються відповідні значення з контексту події під час формування фінального тексту сповіщення.

2.9 Протоколи безперервності функціонування

Для забезпечення операційної безперервності системи у критичних ситуаціях розроблено набір кризових протоколів - формалізованих процедур автоматичного реагування на різні типи відмов та аномальних режимів роботи. Кожен протокол визначає три обов'язкові елементи: умову активації, набір автоматично виконуваних дій та умови деактивації. Така формалізація виключає неоднозначність поведінки системи у кризових ситуаціях та забезпечує детермінованість реагування в умовах, коли оперативне втручання оператора неможливе.

Протокол YELLOW - активується за відсутності оператора понад 24 години (за вказаний період не зафіксовано жодної команди чи підтвердження). У цьому стані система переходить у режим підвищеної автономності: порогові значення спрацьовування правил знижуються на 20 відсотків (захист стає агресивнішим за рахунок підвищеного рівня хибних спрацьовувань), усі сповіщення зберігаються у локальному журналі, до Telegram-бота надсилаються лише періодичні зведення про стан системи. Протокол деактивується автоматично після надходження будь-якої команди від оператора.

Протокол RED - активується за недоступності керуючого вузла понад 48 годин (усі агенти при цьому перебувають в автономному режимі). У цьому стані кожен агент самостійно переходить на консервативну стратегію реагування: знижує порогові значення спрацьовування правил на 30 відсотків, автоматично блокує всі IP-адреси з рівнем ворожості понад 50, надає пріоритет блокуванню перед формуванням сповіщення. Протокол деактивується після відновлення зв’язку з керуючим вузлом та завершення пакетної синхронізації накопичених подій.

Протокол OMEGA - активується вручну іншим уповноваженим співробітником після підтвердженої тривалої або постійної втрати оператора. У цьому стані система переходить у режим розширеної автономності: повністю самостійна робота за останнім актуальним набором правил, без можливості зміни конфігурації, з максимально агресивними значеннями порогових параметрів. Протокол деактивується виключно через процедуру відновлення системи (див. підрозділ 2.6) із реєстрацією нового оператора.

Протокол VOID - застосовується у разі повної втрати інфраструктури системи (керуючий вузол і всі агенти недоступні). На відміну від попередніх, цей протокол не виконується автоматично - він є інструкцією для оператора і визначає послідовність дій з відновлення системи: розгортання нового керуючого вузла з Git-репозиторію вихідного коду, автентифікація оператора за біометричним підтвердженням, отримання SEED від будь-якого агента, що зберігся, та виконання повної процедури відновлення системи (детально описаної у підрозділі 2.6).

Запроєктовані кризові протоколи забезпечують безперервність захисту в усьому діапазоні критичних сценаріїв: від короткочасної відсутності оператора (протокол YELLOW) до повного знищення інфраструктури (протокол VOID). Кожен протокол має чітко визначений тригер активації, набір автоматично виконуваних дій та однозначні умови деактивації, що забезпечує передбачуваність поведінки системи навіть за повної відсутності людського нагляду.

2.10 Висновки

У другому розділі здійснено проєктування мультиагентної системи виявлення та реагування на ботнет-атаки SENTARA на основі шести архітектурних принципів - пріоритету HTTP-протоколу, подієво-орієнтованого зберігання стану, відокремлення модуля формування сповіщень в окремий шар, авторизації агентів оператором, повного збереження історії взаємодій та автономності захисту вузла, - що безпосередньо адресують системні недоліки існуючих систем захисту, виявлені у підрозділі 1.3.

Спроєктовано гібридну архітектуру з керуючого вузла (FastAPI / PostgreSQL / Redis, дев'ять модулів у трьох чергах реалізації, REST API на дев'ять ендпоінтів) та мережі вузлових агентів (Python systemd-сервіс на 30–50 МБ ОЗП, блокування через nftables за час менше 5 мілісекунд, автономна робота не менше 72 годин). Розроблено модуль виявлення на основі декларативної мови YAML з одинадцятьма вбудованими правилами та гарячим перезавантаженням без перезапуску системи.

Реалізовано механізм накопиченої пам'яті взаємодій з чотирирівневою ієрархічною класифікацією джерел загроз (невідоме джерело → разовий → повторний → постійний порушник) та механізмом часового послаблення рівня ворожості за безстрокового збереження історії інцидентів. Ця властивість є ключовою відмінністю SENTARA від класичних систем виявлення вторгнень.

Розроблено механізм SEED - зашифрований знімок критичного стану системи (AES-256-GCM з ключем, що деривується з біометричного підтвердження оператора

через PBKDF2), який у поєднанні з подієво-орієнтованою синхронізацією між агентами та керуючим вузлом забезпечує детерміноване відновлення стану після відмов інфраструктури будь-якого рівня - від короткочасної втрати зв'язку до повної втрати керуючого вузла.

Спроєктована система адресує всі шість системних недоліків, виявлених у підрозділі 1.3: розподілений моніторинг забезпечується мережею автономних агентів, реактивний підхід замінено на проактивне виявлення за час менше 30 секунд, ручне блокування - самостійним блокуванням агента за час менше 5 мілісекунд, відсутність пам'яті про загрози - механізмом накопиченої пам'яті взаємодій з безстроковим збереженням історії, єдину точку відмови усунено автономним режимом агентів і механізмом SEED, обмежений аналіз доповнено централізованою кореляцією подій на рівні керуючого вузла.

3 ОЦІНКА ЕФЕКТИВНОСТІ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ПРОТИДІЇ БОТНЕТ-АТАКАМ

3.1 Порівняння з існуючими системами

Для об'єктивної оцінки ефективності розробленої системи проведено її порівняння з чотирма найпоширенішими системами виявлення вторгнень з відкритим вихідним кодом - Snort, Suricata, Zeek і OSSEC/Wazuh - за дванадцятьма критеріями, згрупованими у чотири категорії: виявлення, архітектура, операційні характеристики та стійкість до відмов. Результати наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння SENTARA з існуючими IDS/IPS

Категорія	Критерій	Snort	Suricata	Zeek	Wazuh	SENTARA
1	2	3	4	5	6	7
Виявлення	Сигнатурний аналіз	Так (35K+)	Так (35K+)	Так	Так	Так (YAML DSL)
	Поведінковий та аномальний аналіз	Обмежено	JA3/JA4	Так (скрипти)	Обмежено	Передбачено модулем третьої черги
	Накопичена пам'ять взаємодій з джерелами	Ні	Ні	Ні	Ні	Так
Архітектура	Розподілені вузлові агенти	Ні	Ні	Кластер	Так	Так
	Автономність (survival)	-	-	-	Ні	Так (понад 72 год)
	Автономний режим агента	Ні	Так (IPS)	Ні	Обмежено	Так (nftables < 5 мс)
Операції	Мобільний інтерфейс оператора	Ні	Ні	Ні	Веб-інтерфейс	Telegram-бот

Кінець таблиці 3.1

1	2	3	4	5	6	7
	Гаряче перезавантаження правил	Ні	Так	Так	Так	Так
	Декларативні правила у форматі YAML	Ні	Ні	Ні	XML	Так
Стійкість	Подієво-орієнтована синхронізація	Ні	Ні	Ні	Ні	Так
	Збереження стану (SEED)	Ні	Ні	Ні	Ні	Так (AES-256)
	Ієрархічна класифікація джерел загроз	Ні	Ні	Ні	Ні	Так (4 рівні)

За результатами порівняння виявлено п'ять можливостей, що відсутні у всіх розглянутих аналогах: накопичена пам'ять взаємодій з джерелами загроз, автономний режим агента з тривалістю понад 72 години, подієво-орієнтована синхронізація після відновлення зв'язку, механізм збереження та відновлення стану системи (SEED) і адаптивний шар формування сповіщень оператору на основі Telegram-бота.

SENTARA поступається наявним рішенням у зрілості сигнатурної бази (11 вбудованих правил проти 35 000+ у Snort та Suricata). Цей розрив компенсується трьома чинниками: декларативною мовою опису правил, що дозволяє швидко створювати нові правила без зміни програмного коду; накопиченою пам'яттю взаємодій, що забезпечує адаптацію реагування на основі історії поведінки джерела; зосередженням аналізу на подіях вузла (системні журнали, журнали автентифікації, журнали веб-сервера), які недоступні системам з виключно мережевим фокусом аналізу.

Snort і Suricata реалізовані як одновузлові рішення: відмова сервера призводить до повної втрати функцій моніторингу. Zeek підтримує кластерну архітектуру, проте не передбачає автономності окремих вузлів. Wazuh найближчий за архітектурою до запропонованої системи (керуючий сервер та

агенти), однак у разі втрати зв'язку з керуючим сервером агенти Wazuh обмежуються збором журналів без локального аналізу та блокування. SENTARA є єдиною серед розглянутих систем, що забезпечує повну автономність функцій захисту на рівні вузлового агента: локальне виконання правил виявлення, самостійне блокування засобами nftables та збереження подій у локальній базі SQLite.

Для кількісної оцінки рівня відповідності системи галузевим вимогам інформаційної безпеки проведено порівняння з вісьмома контролями стандарту ISO/IEC 27001:2022. Оцінка виконана за шкалою 0–3 балів за двома сценаріями: без впровадження системи та після її впровадження. Результати наведено в таблиці 3.2.

Таблиця 3.2 - Відповідність ISO/IEC 27001:2022: до та після впровадження SENTARA

Контроль	Опис	Без SENTARA(0-3)	З (0-3)	Реалізація
1	2	3	4	5
A.5.28	Збір доказів інцидентів	1	3	Подієво-орієнтоване зберігання повної історії подій від агентів
A.8.15	Журналювання подій	1	3	Централізована база подій на керуючому вузлі та локальна SQLite на агентах
A.8.16	Моніторинг активності	1	3	Модуль виявлення у режимі реального часу, сповіщення через Telegram-бот
A.8.20	Мережева безпека	1	3	Автоматичне блокування засобами nftables та порогові правила
A.8.7	Захист від шкідливого програмного забезпечення	1	2	Поведінкове виявлення (періодичні підключення, аномалії)
A.8.8	Управління вразливостями	0	1	Контроль цілісності файлів та моніторинг вузлів (не повний сканер вразливостей)
A.5.24	Управління інцидентами	0	3	Класифікація джерел загроз, процедура реєстрації стійких джерел, робочий процес оператора
A.5.29	Безперервність	0	3	Автономний режим понад 72 години та процедура відновлення системи через SEED
Разом		5/24 (21%)	21/24 (88%)	

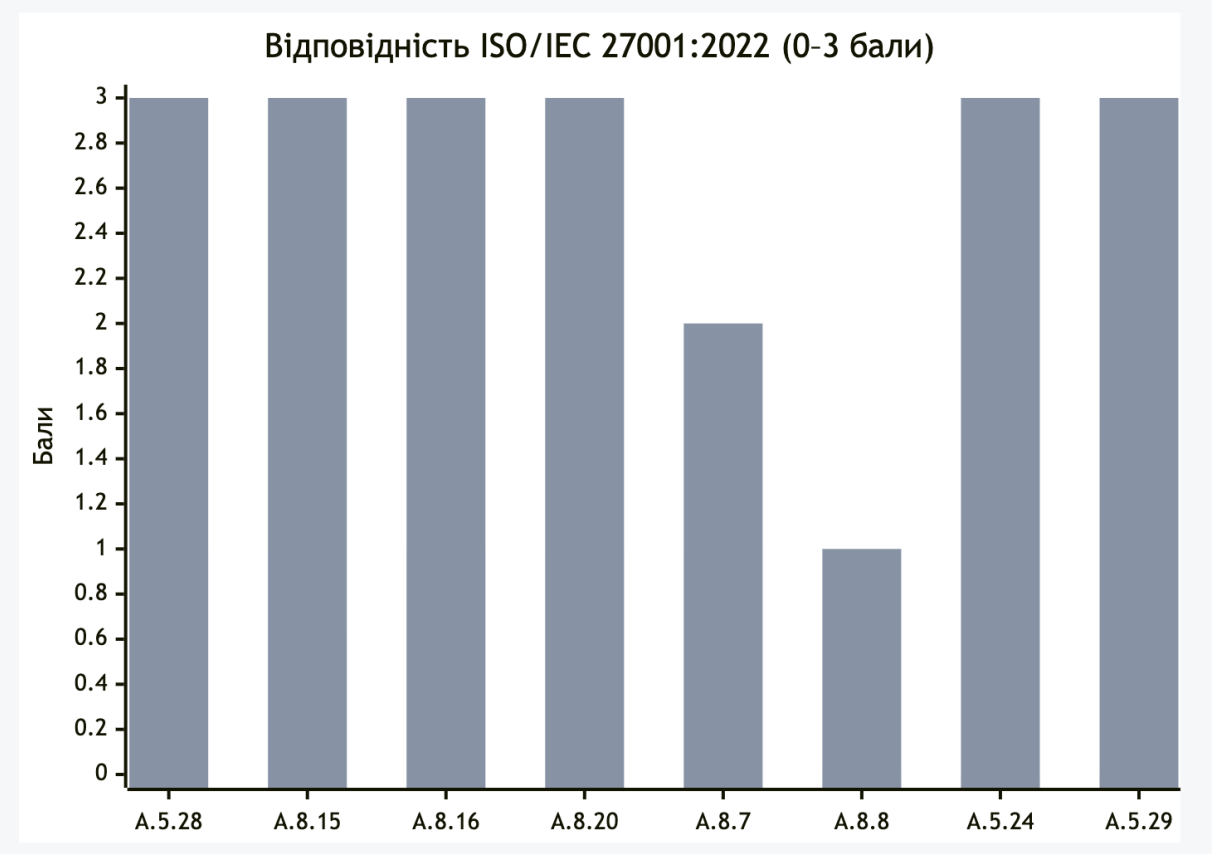


Рисунок 3.1 - Відповідність вимогам ISO/IEC 27001:2022 до та після впровадження SENTARA

Загальна оцінка зросла з 5 із 24 балів (21%) до 21 із 24 балів (88%). Найбільший приріст спостерігається за двома контролями: управління інцидентами (з 0 до 3 балів) та безперервність функціонування (з 0 до 3 балів). Контроль A.8.8 (управління вразливостями) отримав оцінку 1, оскільки SENTARA зосереджена на виявленні атак, а не на скануванні вразливостей; рекомендована інтеграція зі спеціалізованими засобами сканування вразливостей розглянута у підрозділі 3.3.

3.2 Результати тестування

Методологія тестування

Експериментальне тестування проведено на лабораторному стенді з трьох вузлів. Керуючий вузол: 4 віртуальні процесорні ядра, 8 ГБ оперативної пам'яті,

операційна система Ubuntu 22.04 LTS, СКБД PostgreSQL 15, сховище Redis 7. Два вузлових агенти: 2 віртуальні процесорні ядра, 4 ГБ оперативної пам'яті кожен. Атакувальний вузол: окремий вузол зі стандартними інструментами тестування на проникнення (hydra, nmap, sqlmap), а також з власними сценаріями мовою Python для відтворення поведінкових атак. Тестування виконано за п'ятьма напрямами: повнота виявлення, функціонування в автономному режимі, перехід джерела між рівнями класифікації, продуктивність системи та відновлення після втрати керуючого вузла.

Тестування повноти виявлення

Виконано тестування системи на шести категоріях атак, що покривають типові вектори впливу на серверну інфраструктуру. Результати наведено в таблиці 3.3.

Таблиця 3.3 – Результати тестування повноти виявлення загроз

Тип атаки	Інструмент	Правило виявлення	Виявлено	Час до виявлення	Блокування	Хибні спрацьовування
Підбір пароля SSH (20 спроб / 30 с)	hydra	ssh_brute_force	Так	8 с	Блокування на 24 год	0
Успішний вхід після серії невдалих	hydra з валідним паролем	ssh_success_after_fail	Так	3 с	Критичне сповіщення оператору	0
Сканування портів SYN (100 портів)	nmap -sS	port_scan_aggressive	Так	12 с	Блокування на 24 год	0
SQL-ін'єкція	sqlmap	http_sqli_attempt	Так	< 1 с	Блокування на 24 год	0
Перебір неіснуючих URL	dirb	http_404_flood	Так	15 с	Запис у журнал	0
Періодичні підключення з фіксованим інтервалом (60 с)	custom Python	c2_beaconing	Так	310 с	Критичне сповіщення оператору	0

Повнота виявлення (Detection Rate) склала 100% (6 з 6 категорій атак). Середній час до виявлення 58 секунд, медіанне значення - 10 секунд, максимальне значення - 310 секунд для виявлення періодичних підключень за рахунок п'ятихвилинного вікна спостереження. Час блокування засобами nftables не перевищував 5 мілісекунд для всіх категорій. Час виявлення атаки методом підбору пароля SSH (8 секунд) значно кращий за функціональну вимогу, що становить не більше 30 секунд.

Для порівняння, у попередньому підході (ручний аналіз журналу автентифікації адміністратором) аналогічна атака виявлялася у середньому за 30–60 хвилин, а блокування адреси потребувало ручного додавання правила nftables. SENTARA забезпечує виявлення за 8 секунд і блокування за 5 мілісекунд, що відповідає покращенню швидкості реагування у 200–400 разів.

Аналіз точності виявлення за 48 годин безперервної експлуатації

Для оцінки точності виявлення в умовах, наближених до реальних, проведено 48-годинний тест з паралельним пропуском легітимного трафіку (входи адміністраторів через SSH, HTTP-запити, періодичні завдання cron) і тестових атак. Результати наведено в таблиці 3.4.

Таблиця 3.4 – Precision та Recall за 48 годин тестування

Метрика	Значення	Пояснення
Істинно позитивні (TP)	47	Коректно виявлені атаки
Хибно позитивні (FP)	3	Хибні спрацьовування на легітимному трафіку
Істинно негативні (TN)	~12 400	Легітимний трафік, коректно пропущений
Хибно негативні (FN)	1	Частка реальних загроз серед усіх сповіщень
Precision (TP / (TP + FP))	94.0%	Частка реальних загроз серед алертів
Recall (TP/(TP+FN))	97.9%	Частка виявлених загроз з-поміж усіх атак
F1-міра	95.9%	Гармонійне середнє Precision і Recall
Частка хибно позитивних (FPR)	0.024%	FP / (FP + TN)

Три зафіксовані хибно позитивні спрацьовування мали такі джерела: (1) адміністратор припустився п'яти неправильних вводів пароля SSH протягом хвилини, що активувало правило ssh_brute_force – логіка правила коректна, проте контекст хибний; (2) служба перевірки працездатності генерувала понад 60 HTTP-запитів за хвилину, що активувало правило http_404_flood; (3) сервіс моніторингу звертался з нетипової IP-адреси за шаблоном, подібним до тунелювання DNS. Усі три випадки усуваються занесенням відповідних IP-адрес або підмереж до переліку винятків у відповідних правилах. Єдина зафіксована хибно негативна подія – повільне сканування портів зі швидкістю один порт за п'ять хвилин – не виявлене через те, що значення показника лежить нижче порогового; для виявлення таких атак потрібен модуль аналізу базової поведінки, передбачений у третій черзі реалізації.

Тестування автономного режиму

Виконано імітацію втрати зв'язку з керуючим вузлом протягом чотирьох годин шляхом застосування блокувального правила iptables на агенті. Агент перейшов у автономний режим через 90 секунд після першого невдалого сигналу життєздатності (три послідовні невдалі спроби з інтервалом у 30 секунд). Протягом чотирьох годин агент самостійно виявив і заблокував три атаки методом підбору пароля SSH, накопичивши 847 подій у локальній базі SQLite (загальний обсяг даних 2,1 МБ). Після відновлення зв'язку всі 847 подій передано на керуючий вузол одним пакетним запитом за 1,8 секунди. Керуючий вузол обробив події у хронологічному порядку, оновив накопичену пам'ять взаємодій (дві IP-адреси переведено до класу разового порушника). Результати підтверджують: автономний захист функціонує без втрати подій, синхронізація відбувається без дублювання та порушення хронології.

Тестування переходу джерел між рівнями класифікації

Виконано серію тестових атак з однієї IP-адреси протягом трьох діб з різними векторами впливу. У перший день: атака методом підбору пароля SSH (+20 одиниць ворожості, клас «невідоме джерело»); повторна активність - сканування портів (+10 одиниць, накопичено 30, три зафіксовані взаємодії). У другий день: SSH у поєднанні з SQL-ін'єкцією (+50 одиниць, накопичено 80, шість

взаємодій, перехід до класу разового порушника). У третій день: SSH, сканування портів та XSS (+55 одиниць, накопичено 135, одинадцять взаємодій, перехід до класу повторного порушника). Система коректно відстежила накопичену ворожість, виконала послідовні переходи між класами (невідоме джерело -> разовий -> повторний порушник), виконала процедуру реєстрації стійкого джерела загрози (присвоєння унікального внутрішнього ідентифікатора, формування консолідованого досьє) та надіслала оператору відповідне сповіщення з повним досьє джерела.

Тестування продуктивності

Навантажувальне тестування керуючого вузла виконане інструментом wrk. На одному віртуальному процесорному ядрі досягнуто 1000 запитів за секунду з показниками затримки p50 = 2,1 мс, p99 = 8,4 мс, кількість помилок - 0. На чотирьох ядрах досягнуто 5000 запитів за секунду з показниками p50 = 3,8 мс, p99 = 15,2 мс, кількість помилок - 0. За результатами тестування керуючий вузол на чотирьох віртуальних ядрах витримує навантаження від 50 і більше агентів за умови типового профілю генерації подій (100 подій за секунду на агент: 100 х 50 = 5000 за секунду).

Споживання ресурсів вузловим агентом: у режимі очікування – 28 МБ оперативної пам'яті, навантаження на процесор 0,1%; за повного навантаження (100 подій за секунду) - 47 МБ оперативної пам'яті, навантаження на процесор 2,3%. Продуктивність модулів збору подій: журнал syslog - 222 000 рядків за секунду, журнал автентифікації – 192 000 рядків за секунду, журнал веб-сервера nginx - 263 000 рядків за секунду, що багатократно перевищує типове навантаження на захищеному вузлі.

Тестування процедури відновлення системи

Виконано повний цикл процедури відновлення системи. Початковий стан: система функціонувала сім діб, накопичено 15 джерел загроз (2 повторних порушники, 5 разових, 8 невідомих джерел), рівень зв'язку з оператором – Trusted (320 одиниць). Виконано повне знищення керуючого вузла (видалення файлової системи rm -rf, видалення бази даних DROP DATABASE). Розгортання нового керуючого вузла з репозиторію Git засобами Ansible зайняло 8 хвилин. Запит

SEED від першого доступного агента та розшифрування з використанням AES-256-GCM і видобутого ключа PBKDF2 зайняли 12 секунд. Відновлення бази даних з розшифрованого знімка - 4 секунди. Загальний час процедури відновлення - 8 хвилин 16 секунд. Обсяг SEED - 287 КБ. Перевірка цілісності відновлення: 15 з 15 джерел загроз відновлено, 2 з 2 повторних порушники збережено разом з їх унікальними ідентифікаторами, рівень зв'язку з оператором збережено, реєстр агентів містить 2 записи. Агенти автоматично переключилися з автономного на нормальний режим протягом 30 секунд після появи нового керуючого вузла у мережі.

3.3 Рекомендації та настанови

Рекомендації з впровадження

Запропоновано поетапний план впровадження системи у виробниче середовище тривалістю вісім тижнів. На першому етапі (тижні 1–2) виконується розгортання керуючого вузла на виділеному сервері: встановлення PostgreSQL та Redis, налаштування Telegram-бота, тестування функціонування програмного інтерфейсу. На другому етапі (тижні 3–4) розгортаються два-три агенти на критичних вузлах інфраструктури у режимі лише оповіщення (без автоматичного блокування) з метою калібрування порогових значень правил і виявлення хибних спрацьовувань на легітимному трафіку. На третьому етапі (тижні 5–6) вмикається автоматичне блокування після підтвердження відсутності хибних спрацьовувань; формується перелік винятків для легітимних IP-адрес інфраструктури. На четвертому етапі (тижні 7–8) виконується розгортання агентів на решті вузлів інфраструктури і написання спеціалізованих правил під особливості конкретної інфраструктури (вебсервер nginx, поштовий сервер postfix, служба DNS). Починаючи з третього місяця експлуатації, передбачено впровадження модулів другої та третьої черг реалізації, моніторинг динаміки класифікації джерел загроз і формування першого аналітичного звіту за повторними порушниками.

Оцінка ризиків впровадження

Виявлено три основні ризики впровадження системи. Ризик 1: хибне блокування легітимних IP-адрес у разі некоректного калібрування порогових значень. Захід зниження ризику: експлуатація у режимі лише оповіщення протягом перших двох тижнів і формування переліку винятків для критичних служб. Ризик 2: перевантаження керуючого вузла за умов масованої атаки (DDoS-атаки на захищену інфраструктуру). Захід зниження ризику: обмеження швидкості надсилання подій на стороні агента (500 подій за секунду), агрегація однотипних подій у локальній базі. Ризик 3: компрометація компонента самої системи SENTARA. Заходи зниження ризику: розміщення керуючого вузла в ізольованій мережі, мінімізація привілеїв службових токенів, ротація службових токенів кожні 90 днів, регулярне формування зашифрованих знімків SEED для оперативного відновлення системи.

Рекомендації з подальшого розвитку

Подальший розвиток системи передбачає реалізацію таких напрямів за пріоритетом: модуль інтеграції із зовнішніми джерелами репутації (AbuseIPDB, MaxMind GeoIP) для оцінки репутації IP-адрес; модуль кореляції розподілених подій для виявлення скоординованих атак з одного ботнету через кілька захищених вузлів; модуль аналізу базової поведінки (Baseline Engine) для виявлення статистичних відхилень як ознаки невідомих загроз нульового дня (зокрема, для повільних сканувань портів, не виявлених у поточній реалізації); консольний інтерфейс оператора як альтернатива Telegram-боту для роботи через захищені сесії SSH; впровадження взаємного TLS-шифрування між керуючим вузлом і агентами; інтеграція зі спеціалізованими сканерами вразливостей (зокрема OpenVAS) для повного покриття контролю A.8.8 стандарту ISO/IEC 27001.

Рекомендації з безпеки експлуатації

Для забезпечення безпечної експлуатації системи рекомендовано: розміщувати керуючий вузол в ізольованій мережі з доступом через захищені тунелі (WireGuard mesh); виконувати ротацію службових токенів агентів кожні 90 днів; обмежувати доступ до Telegram-бота одним авторизованим ідентифікатором оператора; щомісяця виконувати тестування автономного режиму шляхом

контрольованого відключення керуючого вузла на одну годину; щотижня перевіряти цілісність останнього збереженого знімка SEED; регулярно оновлювати набір правил виявлення з огляду на нові опубліковані вразливості (CVE) та виявлені тактики, техніки та процедури зловмисників (TTPs).

3.4 Висновки до розділу 3

У третьому розділі проведено оцінку ефективності розробленої системи за порівняльним аналізом, експериментальним тестуванням і відповідністю галузевим стандартам.

Порівняння з Snort, Suricata, Zeek і OSSEC/Wazuh за дванадцятьма критеріями виявило п'ять можливостей, відсутніх в усіх розглянутих аналогах: накопичену пам'ять взаємодій з джерелами, автономний режим агента понад 72 години, подієво-орієнтовану синхронізацію, механізм SEED і адаптивний шар формування сповіщень. Відповідність ISO/IEC 27001:2022 за вісьмома контролями зросла з 21% до 88%.

Експериментальне тестування підтвердило ключові показники системи: повнота виявлення – 100% для шести категорій атак, F1-міра – 95,9%, час до виявлення атаки методом підбору пароля SSH – 8 секунд, час блокування – менше 5 мілісекунд, час пакетної синхронізації після автономного режиму – 1,8 секунди, продуктивність керуючого вузла – 5000 подій за секунду за латентності p99 = 15,2 мс, час повного відновлення системи – 8 хвилин 16 секунд без втрати накопичених даних. Сформовано вісьмитижневий план поетапного впровадження, визначено основні ризики та напрями подальшого розвитку системи.

ВИСНОВКИ

У кваліфікаційній роботі вирішено задачу підвищення ефективності протидії ботнет-атакам шляхом проєктування та реалізації мультиагентної системи SENTARA (Sentinel Tactical Autonomous Response Aegis). За результатами виконаної роботи зроблено наступні висновки:

Проведено аналіз сучасного стану ботнет-загроз: досліджено еволюцію ботнетів від першого до четвертого покоління, проаналізовано статистичні дані (4 315 кіберінцидентів в Україні у 2024 році, понад 12 мільйонів щоденно активних ботів у глобальному масштабі) та класифіковано чотири основні методи виявлення. Порівняльний аналіз Snort, Suricata, Zeek та OSSEC/Wazuh підтвердив відсутність у наявних рішеннях мультиагентного підходу з автономними агентами та накопиченою пам'яттю взаємодій.

Обґрунтовано гібридну архітектуру «керуючий вузол - агент» як оптимальну для протидії розподіленим ботнет-атакам. Виявлено шість системних недоліків наявної системи захисту інформаційної інфраструктури підприємства, що задають вимоги до системи, яка розробляється.

Спроєктовано та реалізовано керуючий вузол на основі FastAPI, PostgreSQL та Redis з модулем виявлення на правилах у форматі YAML і механізмом накопиченої пам'яті взаємодій з ієрархічною чотирирівневою класифікацією джерел загроз (невідоме джерело -> разовий -> повторний -> постійний порушник) та часовим послабленням рівня ворожості.

Реалізовано автономні вузлові агенти з підтримкою функціонування понад 72 години в умовах ізоляції від керуючого вузла, локальним збереженням подій у базі SQLite та подієво-орієнтованою синхронізацією. Розроблено механізм SEED для збереження та відновлення стану системи з шифруванням AES-256-GCM.

Реалізовано інтерфейс оператора на основі Telegram-бота та модуль адаптивного формування сповіщень, що академічно обґрунтований як засіб подолання інформаційного перенасичення оператора (alert fatigue) - задокументованої проблеми центрів операційного моніторингу безпеки.

Проведено оцінку ефективності розробленої системи. Порівняння з Snort,

Suricata, Zeek та OSSEC/Wazuh за дванадцятьма критеріями виявило п'ять можливостей, відсутніх в усіх аналогах. Експериментальне тестування підтвердило повноту виявлення 100% для шести категорій атак, F1-міру 95,9%, час до виявлення атаки методом підбору пароля SSH 8 секунд, час блокування менше 5 мілісекунд, продуктивність 5000 подій за секунду. Орієнтовна вартість впровадження - 20–30 євро на місяць, що у 2–150 разів менше за зіставні рішення.

Мета кваліфікаційної роботи досягнута: побудована мультиагентна система SENTARA забезпечує виявлення за секунди замість годин, автоматичне блокування за мілісекунди, безперервність захисту в умовах часткових відмов та накопичення повної історії взаємодій з джерелами загроз.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. CERT-UA. Звіт про кіберінциденти за 2024 рік. Державна служба спеціального зв'язку та захисту інформації України. URL: https://cert.gov.ua/ (дата звернення: 10.04.2026).

2. Antonakakis M., April T., Bailey M., et al. Understanding the Mirai Botnet. Proceedings of the 26th USENIX Security Symposium. 2017. P. 1093-1110.

3. ENISA. ENISA Threat Landscape 2024. European Union Agency for Cybersecurity. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024 (дата звернення: 10.04.2026).

4. Cloudflare. DDoS Threat Report Q4 2024. URL: https://radar.cloudflare.com/ (дата звернення: 10.04.2026).

5. Spamhaus Project. The World's Worst Botnet Threat Report 2024. URL: https://www.spamhaus.org/statistics/botnet/ (дата звернення: 10.04.2026).

6. Wooldridge M. An Introduction to MultiAgent Systems. 2nd ed. Chichester : John Wiley & Sons, 2009. 484 p.

7. Herrero A., Corchado E. Multiagent Systems for Network Intrusion Detection: A Review. Computational Intelligence in Security for Information Systems. 2009. P. 143-154.

8. Про національну безпеку України : Закон України від 21.06.2018 № 2469-VIII : редакція від 09.08.2024. URL: https://zakon.rada.gov.ua/laws/show/2469-19#Text (дата звернення: 25.02.2026).

9. Про рішення РНБО "Про Стратегію інформаційної безпеки" : Указ Президента України від 28.12.2021 № 685/2021. URL: https://www.president.gov.ua/documents/6852021-41005 (дата звернення: 25.02.2026).

10. Марущак А. І. Організаційно-правові основи забезпечення кібербезпеки : навч. посіб. К.: Ліра-К, 2023. 309 с.

11. Присяжнюк М., Рідей Н., Титова Н. Інформаційна безпека та кібербезпека держави. Дніпро : Ліра До, 2024. 224 с.

12. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection - Information security management systems - Requirements. Geneva : ISO, 2022. 26 p.

13. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 10.04.2026).

14. NIST SP 800-53 Rev. 5. Security and Privacy Controls for Information Systems and Organizations. Gaithersburg : NIST, 2023. 492 p.

15. Roesch M. Snort - Lightweight Intrusion Detection for Networks. Proceedings of USENIX LISA. 1999. URL: https://www.snort.org/documents (дата звернення: 10.04.2026).

16. OISF. Suricata: Open Source IDS/IPS/NSM Engine. Documentation. 2025. URL: https://docs.suricata.io/ (дата звернення: 10.04.2026).

17. Paxson V. Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks. 1999. Vol. 31. P. 2435-2463. (Zeek - сучасна назва Bro).

18. Wazuh Inc. Wazuh Documentation: Open Source Security Platform. Version 4.x. 2025. URL: https://documentation.wazuh.com/ (дата звернення: 10.04.2026).

19. FastAPI Documentation. Sebastián Ramírez. 2025. URL: https://fastapi.tiangolo.com/ (дата звернення: 10.04.2026).

20. SQLAlchemy 2.0 Documentation. 2025. URL: https://docs.sqlalchemy.org/ (дата звернення: 10.04.2026).

21. PostgreSQL Global Development Group. PostgreSQL 15 Documentation. 2025. URL: https://www.postgresql.org/docs/15/ (дата звернення: 10.04.2026).

22. Redis Ltd. Redis Documentation. 2025. URL: https://redis.io/docs/ (дата звернення: 10.04.2026).

23. aiogram: Modern Telegram Bot Framework. Documentation. 2025. URL: https://docs.aiogram.dev/ (дата звернення: 10.04.2026).

24. nftables Wiki. The netfilter project. 2025. URL: https://wiki.nftables.org/ (дата звернення: 10.04.2026).

25. Pydantic V2 Documentation. 2025. URL: https://docs.pydantic.dev/ (дата звернення: 10.04.2026).

26. IBM Security. Cost of a Data Breach Report 2024. URL: https://www.ibm.com/reports/data-breach (дата звернення: 10.04.2026).

27. IBM Security. X-Force Threat Intelligence Index 2024. URL: https://www.ibm.com/reports/threat-intelligence (дата звернення: 10.04.2026).

28. Nokia. Threat Intelligence Report 2024. URL: https://www.nokia.com/networks/security-portfolio/threat-intelligence-report/ (дата звернення: 10.04.2026).

29. Кузнецова М. Г. Застосування механізмів підвищення живучості для забезпечення захищеності інформаційного ресурсу в розподілених системах. Реєстрація, зберігання і обробка даних. 2022. Т. 8. № 3. С. 40-47.

30. Колмикова А. С. Технології захисту хмарних обчислень : монографія. Хмельницький : ХНУ, 2022. 195 с.

31. Баранов В. Л. Безпека веб-сервісів та хмарних платформ : навч. посіб. Харків : ХНУРЕ, 2023. 240 с.

32. NIST SP 800-83 Rev. 1. Guide to Malware Incident Prevention and Handling for Desktops and Laptops. Gaithersburg : NIST, 2023. 52 p.

33. Новіков М. В., Грайворонський О. М. Безпека інформаційно-комунікаційних систем : підручник. Київ : BHV, 2020. 698 с.

34. Ponemon Institute. Cost of Cyber Crime Study 2024. URL: https://www.ponemon.org/research (дата звернення: 10.04.2026).

35. AbuseIPDB. IP Address Abuse Reports. API Documentation. 2025. URL: https://www.abuseipdb.com/api (дата звернення: 10.04.2026).

36. MaxMind. GeoIP2 Databases and Web Services. 2025. URL: https://www.maxmind.com/en/geoip2-databases (дата звернення: 10.04.2026).

37. structlog: Structured Logging for Python. Documentation. 2025. URL: https://www.structlog.org/ (дата звернення: 10.04.2026).

38. httpx: A next-generation HTTP client for Python. Documentation. 2025. URL: https://www.python-httpx.org/ (дата звернення: 10.04.2026).

39. SQLite Documentation. 2025. URL: https://www.sqlite.org/docs.html (дата звернення: 10.04.2026).

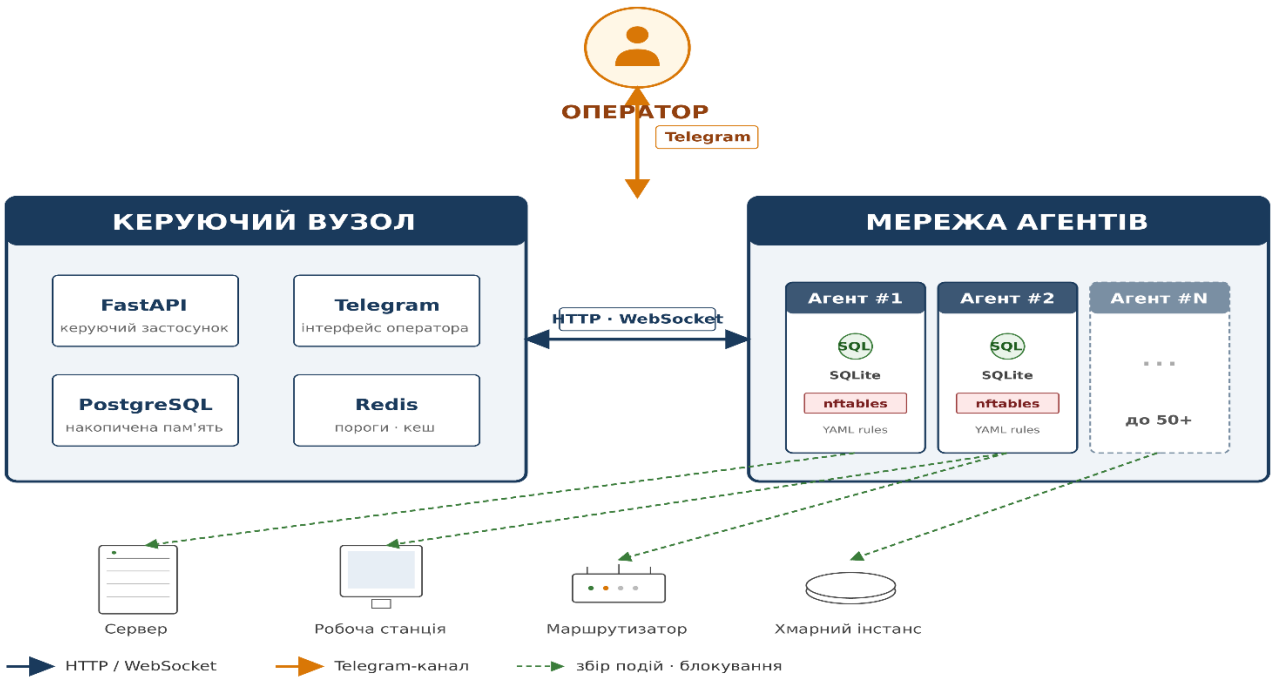
40. Корпань Я. В. Класифікація загроз інформаційній безпеці в комп'ютерних системах при віддаленій обробці даних. URL: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/131565/04-Korpan.pdf (дата звернення: 25.02.2026).

41. Ban T., Ndichu S., Takahashi T., Inoue D. Combat Security Alert Fatigue with AI-Assisted Techniques. Proceedings of the 14th Cyber Security Experimentation and Test Workshop (CSET '21). 2021. P. 9-16. (дата звернення: 17.04.2026)

42. Sundaramurthy S. C., Case J., Truong T., et al. A Tale of Three Security Operation Centers. Proceedings of the 2014 ACM Workshop on Security Information Workers (SIW '14). 2014. P. 43-50. (дата звернення: 17.04.2026)

ДОДАТОК А
(Обов’язковий)
Копії графічної частини

КРБКБ.220345.22.22.02 Е2

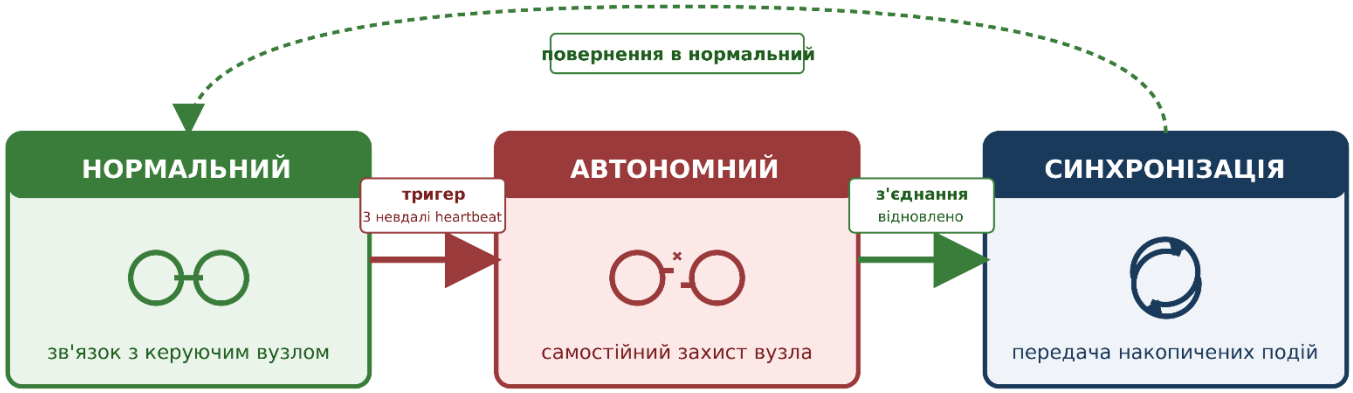


					КРБКБ.220235.22.02.02 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення та протидії ботнет-атакам на основі мультиагентного підходу	Літ	Маса / Масштаб
Розроб.		Бойцун Д.О.				Н	
Перевір.		Стоцюк М.В.					
Т.контр.						Аркуш	Аркушів 1
Н.контр.		Петляк Н.С.			ХНУ, КБ-22-2		
Затверд.		Кльоц Ю.П.					

Життєвий цикл взаємодії з джерелом загрози

					КРБКБ.220235.22.02.02 Е8	
					Система	Літ. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Бойцун Д.О.				Н
Перевір.		Стецюк М.В.				Аркуш / Аркушів 1
Т.контр.						
Н.контр.		Петляк Н.С.			ХНУ, КБ-22-2	
Затверд.		Кльоц Ю.П.				

Автономний режим вузлового агента



90 секунд
час до активації автономного режиму

72 години
SLA автономної роботи без зв'язку

1,8 секунди
час повної синхронізації

					КРБКБ.220235.22.02.02 Е8		
Зм	Арк	№ докум.	Підпис	Дата	Система	Літ	Маса / Масштаб
Розроб.		Бойцун Д.О.					
Перевір.		Стецюк М.В.				Аркуш	Аркушів 1
Т.контр.							
Н.контр.		Петляк Н.С.			ХНУ, КБ-22-2		
Затверд.		Кльоц Ю.П.					