Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Комп'ютерна система захисту конфіденційної інформації від витоків
Назва теми

Рівень виої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 2302139.23.02.29. ПЗ

Виконав здобувач ІІІ курсу, група КІ2с-23-2 _____ Дмитро СЛОБОДЯНЮК
Підпис Ініціали, прізвище

Керівник _____ Олег САВЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«02» червня 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Слободянюку Дмитру Олександровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Комп'ютерна система захисту конфіденційної інформації від витоків

Керівник проекту (роботи) Савенко Олег Станіславович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

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

Комп'ютерна система захисту конфіденційної інформації від витоків та постановка задачі щодо розробки даної системи.

Проєктування архітектури розподіленої системи запобігання витокам даних та математичне обґрунтування багатопотокового моніторингу кінцевих точок.

Програмна реалізація комп'ютерної система захисту конфіденційної інформації від витоків

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура DLP системи

Структура і склад ПЗ комп'ютерної системи

Блок-схеми алгоритмів функціонування системи

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – теоретичні основи досліджуваної проблеми	01.03.2026	виконано
4	Робота над розділом 2 – проєктування та архітектура розподіленої системи захисту інформації від витоків	01.04.2026	виконано
5	Робота над розділом 3 – АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	24.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач_____ Дмитро СЛОБОДЯНЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ Олег САВЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 2302139.23.02.29 ПЗ	Пояснювальна записка	59		
			Графічні матеріали			
2		КвРКІ 2302139.23.02.29 Е8	Архітектура DLP системи	1		
3		КвРКІ 2302139.23.02.29 Е8	Структура і склад ПЗ	1		
			комп'ютерної системи			
4		КвРКІ 2302139.23.02.29 Е8	Блок-схеми алгоритмів	1		
			функціонування системи			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 2302139.23.02.29 ВП		
Розробив		Слободянюк		02.06	Відомість проекту	Літера	Аркуш / Аркушів
Перевір.		Савенко		02.06		У	1 / 1
Н. контр.		Лисенко		02.06		ХНУ, КІ2с-23-2	
Затв.		Павлова		02.06			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Комп'ютерна система захисту конфіденційної інформації від витоків».

Автор роботи: Дмитро СЛОБОДЯНЮК.

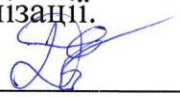
Керівник роботи: Олег САВЕНКО.

Пояснювальна записка: 59 с., 19 рис., 4 дод., 40 джерел.

Графічна частина: 3 креслення.

АВТОНОМНІСТЬ, БАЗА ДАНИХ, БАГАТОПОТОКОВІСТЬ, ВИТІК ІНФОРМАЦІЇ, КІБЕРБЕЗПЕКА, МЕРЕЖЕВИЙ ШЛЮЗ, ПОЛІТИКА БЕЗПЕКИ, РОЗПОДІЛЕНА СИСТЕМА.

Кваліфікаційна робота бакалавра присвячена проєктуванню та програмній реалізації розподіленої універсальної системи запобігання витокам інформації. Актуальність теми зумовлена зростанням інсайдерських загроз та неконтрольованим використанням штучного інтелекту в умовах розмиття корпоративного мережевого периметра. Периферійний моніторинг хостових операцій та інспекція трафіку на рівні оперативної пам'яті дозволяють ефективно попереджати витоки й підвищувати рівень захищеності підприємства. Метою роботи є створення відмовостійкого комплексу для збору, семантичного оброблення та візуалізації інцидентів безпеки в реальному часі. Для її досягнення спроєктовано трирівневу архітектуру на базі периферійних обчислень, розроблено алгоритми паралельного моніторингу кінцевих точок і фільтрації мережевих потоків, реалізовано базу даних, веб-інтерфейс управління та проведено тестування розробленого ПЗ. Окрему увагу приділено створенню превентивних засобів протидії, включаючи механізми низькорівневого затирання файлів та зворотного стирання тексту в режимі реального часу. Результати експериментальних випробувань підтвердили надійність захисту та здатність архітектури до логічної самоорганізації.



Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

Актуальність дослідження зумовлена розмиттям корпоративного мережевого периметра через хмарні технології та гібридну роботу, що робить дані вразливими до інсайдерських загроз і неконтрольованого використання штучного інтелекту. Існуючі закордонні DLP-системи є дорогими, архітектурно складними та погано адаптованими до специфіки української мови. Метою роботи є проєктування та програмна реалізація розподіленої системи запобігання витокам інформації на основі багатопотокового аналізу кінцевих точок і мережевої інспекції для проактивного захисту даних. Об’єктом дослідження є механізми обробки інформації в комп’ютерних системах захисту, а предметом - архітектурні моделі та алгоритми моніторингу локальних і мережевих каналів витоку.

Досягнення мети передбачає вирішення таких задач: аналіз сучасних DLP-засобів, проєктування відмовостійкої трирівневої архітектури, розробка алгоритмів паралельного моніторингу, затирання даних та зворотного стирання тексту, а також програмна реалізація комплексу, створення веб-панелі управління та тестування ПЗ. Методологічну основу складають теорія систем, периферійні обчислення, багатопотокове та подійно-орієнтоване програмування, архітектури MVC і REST, а також методи семантичного аналізу контенту. Практичне значення результатів полягає у розробці кросплатформеного комплексу комплексу захисту даних корпоративного рівня, здатного надійно контролювати канали витоку та легко інтегруватися в інфраструктуру організацій для захисту конфіденційної інформації без значних фінансових витрат.

1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Сучасна ера цифрової трансформації характеризується безпрецедентною інтенсивністю обміну даними, що перетворює конфіденційну інформацію на найбільш вразливий та стратегічно важливий актив будь-якої організації. Станом на 2026 рік предметна область захисту від витоків інформації пройшла шлях від простої периметральної оборони до складних багаторівневих систем, інтегрованих у кожну ланку бізнес-процесу. Глобальна динаміка кіберзагроз демонструє стрімке зростання фінансових та репутаційних збитків, спричинених несанкціонованим розповсюдженням даних. Це зумовлено не лише професіоналізацією кіберзлочинності, а й складністю сучасних інформаційних ландшафтів, де межа між корпоративним середовищем та особистим простором користувача практично нівельована, що підтверджується актуальною статистикою провідних аналітичних центрів [1,11].

Фундаментальною проблемою предметної області є ерозія класичного поняття мережевого контуру. У попередні десятиліття стратегія безпеки базувалася на створенні жорсткого бар'єра на межі внутрішньої мережі, проте масове впровадження хмарних обчислень та перехід до гібридних моделей роботи змінили правила гри. Сьогодні конфіденційна інформація перебуває у постійному русі, мігруючи між локальними серверами, публічними хмарними сховищами та мобільними пристроями співробітників. Таке розпорошення активів робить традиційні засоби контролю на рівні шлюзів недостатніми, оскільки значна частина трафіку тепер проходить поза межами прямого спостереження корпоративних систем безпеки, що вимагає переосмислення підходів до моніторингу кінцевих точок [7, 14].

Особливої уваги заслуговує трансформація внутрішніх загроз, які на сьогодні визнані найбільш критичним вектором витоку. Статистичні дані останніх років вказують на те, що більшість інцидентів ініціюються суб'єктами,

які мають легітимні права доступу до систем. При цьому важливо розрізняти навмисну крадіжку інтелектуальної власності та ненавмисні помилки персоналу, спричинені низькою обізнаністю або прагненням оптимізувати робочі процеси за допомогою несанкціонованих інструментів [2, 19]. У контексті національної безпеки проблема витоків набуває ще більшої гостроти, оскільки викрадені дані можуть використовуватися для стратегічного тиску або дестабілізації об'єктів критичної інфраструктури. Це диктує необхідність розробки систем, здатних не просто фіксувати факт передачі файлу, а й розуміти контекст та намір користувача в режимі реального часу, спираючись на інтелектуальні алгоритми аналізу [3, 9].

Технологічний розвиток останніх років приніс не лише нові інструменти захисту, а й нові складні виклики, серед яких найбільш значущим є поширення систем генеративного штучного інтелекту. Використання великих мовних моделей для автоматизації рутинних завдань створило прихований канал витоку, де конфіденційний програмний код або стратегічні плани потрапляють у зовнішні бази даних під час формування запитів. Традиційні методи аналізу контенту, засновані на пошуку за ключовими словами або регулярними виразами, виявляються безсилими перед такими сценаріями, оскільки інформація може бути перефразована або змінена до невпізнання. Це породжує завдання створення систем семантичного аналізу, які здатні ідентифікувати чутливу інформацію за її змістом, незалежно від форми представлення [4, 8].

Паралельно з цим спостерігається криза ефективності існуючих рішень через високий рівень хибнопозитивних спрацювань. Величезний потік сповіщень призводить до перевантаження аналітиків безпеки, що своєю чергою спричиняє ігнорування реальних загроз. Сучасна предметна область вимагає переходу до автоматизованого прийняття рішень на основі інтелектуального аналізу поведінки. Такий підхід передбачає побудову математичних моделей нормальної активності для кожного об'єкта та суб'єкта системи, що дозволяє

виявляти аномалії, які можуть свідчити про підготовку до витоку інформації ще на етапі збору даних зловмисником [12, 17].

Важливим аспектом є нормативно-правове регулювання, яке стає все більш суворим. Законодавчі вимоги щодо захисту персональних даних та конфіденційної інформації вимагають від організацій не лише наявності технічних засобів, а й повної прозорості процесів обробки даних [6]. Це ставить перед розробниками завдання забезпечення доказовості та цілісності журналів аудиту, а також необхідність інтеграції систем захисту з процесами управління ризиками. Міжнародні стандарти дедалі частіше пропонують концепцію нульової довіри, де кожен запит на доступ до даних має проходити безперервну верифікацію, незалежно від місця перебування ініціатора [10, 20].

Проблема захисту ускладнюється використанням наскрізного шифрування в корпоративних комунікаціях та месенджерах. Хоча шифрування забезпечує приватність, воно створює сліпі зони для систем моніторингу трафіку. Це змушує переносити функції аналізу безпосередньо на кінцеві точки, де дані перебувають у відкритому вигляді. Проте встановлення агентів на робочі станції часто стикається з проблемами продуктивності та конфліктами з прикладним програмним забезпеченням, що вимагає розробки максимально оптимізованих алгоритмів перехоплення та аналізу системних викликів, які б не перешкоджали стабільній роботі користувача [13, 18].

Окрему увагу в аналізі предметної області слід приділити феномену тіньового IT, коли співробітники самостійно використовують хмарні сервіси для обміну великими обсягами даних. Це призводить до втрати контролю над життєвим циклом інформації, оскільки вона виходить за межі керованої інфраструктури. Сучасна система захисту повинна мати інструменти для виявлення та контролю таких сервісів, забезпечуючи при цьому баланс між безпекою та продуктивністю праці [5, 15]. Це вимагає впровадження брокерів безпечного доступу до хмари та глибокої інтеграції з програмними інтерфейсами найбільш популярних сервісів спільної роботи.

Ще одним викликом є обробка неструктурованих даних, таких як зображення, аудіо- та відеофайли. Велика частка конфіденційної інформації сьогодні передається у вигляді сканів документів або скриншотів, що робить текстовий пошук неефективним. Завданням для нових систем захисту є впровадження методів комп'ютерного зору та автоматизованого розпізнавання об'єктів у реальному часі, що дозволило б класифікувати візуальний контент без суттєвих затримок. Необхідність відстеження походження даних та їх трансформації стає критичною для ідентифікації прихованих каналів витоку [16]

Підсумовуючи аналіз, можна визначити ключовий перелік завдань, що стоять перед дослідниками: розробка методів точної класифікації контенту в умовах використання генеративного штучного інтелекту, створення адаптивних моделей поведінкового аналізу для виявлення інсайдерських загроз, забезпечення контролю даних у мультихмарних середовищах та мінімізація негативного впливу засобів захисту на бізнес-процеси. Тільки комплексний підхід, що поєднує технологічні інновації з глибоким аналізом патернів порушень та вимог законодавства, дозволить створити ефективну систему захисту, здатну протистояти викликам 2026 року. Виявлені проблеми вказують на те, що сигнатурні та статичні методи вичерпали свій потенціал, і подальший розвиток галузі неможливий без широкого впровадження інтелектуальних засобів аналізу та концепції безперервної перевірки довіри до кожного суб'єкта інформаційних відносин.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Еволюція комп’ютерних систем захисту інформації призвела до формування складної ієрархії технологічних рішень, які сьогодні класифікуються не лише за архітектурою, а й за рівнем інтеграції штучного інтелекту та методами обробки великих даних.[25] Станом на 2026 рік основним критерієм ефективності систем запобігання витокам вважається їхня здатність

забезпечувати наскрізну видимість інформаційних потоків у гетерогенних середовищах, що відповідає сучасним вимогам міжнародних стандартів інформаційної безпеки. Аналіз існуючих рішень демонструє чіткий поділ на консервативні системи глибокої перевірки контенту, хмарно-орієнтовані сервіси та платформи поведінкової аналітики, кожна з яких має специфічні переваги та критичні недоліки в контексті сучасного ландшафту загроз.[26, 28]

Одним із найбільш технологічно зрілих рішень на ринку залишається Symantec Data Loss Prevention, яке протягом десятиліть формувало стандарти галузі. Його ключовою перевагою є безпрецедентна потужність математичних алгоритмів ідентифікації контенту, зокрема технологія точного збігу даних, яка дозволяє системі розпізнавати конфіденційні записи з баз даних обсягом у сотні мільйонів рядків без суттєвої деградації продуктивності. Проте аналіз практичного використання цієї системи виявляє її головний недолік - надмірну архітектурну складність. Розгортання та підтримка Symantec DLP потребує значних витрат на апаратне забезпечення та утримання штату висококваліфікованих адміністраторів, що в умовах дефіциту кадрів у сфері кібербезпеки стає серйозною перешкодою для багатьох організацій. Крім того, незважаючи на вдосконалення методів розпізнавання, система залишається чутливою до високого рівня помилкових спрацювань при обробці неструктурованого контенту, що суперечить сучасним підходам до автоматизації безпеки, описаним у наукових працях щодо інтелектуального аналізу даних [23, 40].

На противагу традиційним архітектурам, екосистемний підхід, втілений у Microsoft Purview DLP, демонструє переваги за рахунок глибокої інтеграції в офісні додатки та операційні системи Windows. Перевагою цього рішення є відсутність потреби у встановленні сторонніх агентів на кінцеві точки, що значно знижує ризик виникнення конфліктів із прикладним програмним забезпеченням та спрощує процес оновлення систем. Використання єдиних міток конфіденційності дозволяє організації реалізувати наскрізну політику безпеки

від моменту створення документа до його архівації. Проте детальне порівняння виявляє суттєві обмеження Purview у частині контролю за межами екосистеми Microsoft. Система демонструє недостатню видимість операцій у сторонніх хмарних сховищах та на пристроях під управлінням альтернативних операційних систем, що створює значні сліпі зони у загальній стратегії захисту та не дозволяє повною мірою реалізувати вимоги концепції нульової довіри, яка просувається агенціями з кібербезпеки як єдино можлива модель у сучасному світі [31, 38].

Важливим кроком у розвитку засобів захисту стала поява рішень з адаптивним управлінням ризиками, яскравим представником яких є Forcepoint DLP. Перевага даного підходу полягає у відмові від статичних правил на користь динамічного ранжування подій залежно від показника ризику конкретного користувача. Це дозволяє автоматично пом'якшувати обмеження для перевірених співробітників та посилювати аудит при виявленні підозрілої активності, що відповідає положенням стандарту NIST SP 1800-35. Недоліком Forcepoint залишається складність математичного апарату оцінки поведінки, який часто потребує тривалого періоду навчання на реальних даних організації, що в умовах динамічних бізнес-процесів може призводити до пропуску реальних атак на початкових етапах впровадження. Також фахівці відзначають певну інертність системи при обробці великих масивів зашифрованого трафіку, що стає критичним фактором при роботі з потоковими даними [22, 29].

У сегменті хмарно-орієнтованих систем захисту, таких як Netskope та Zscaler, акцент зміщений на контроль даних у транзиті між хмарними сервісами. Їхньою головною перевагою є здатність забезпечувати захист за межами корпоративної мережі, що є критично важливим для організацій з великою кількістю віддалених працівників. Ці рішення ефективно протидіють витокам через особисті хмарні акаунти та месенджери, використовуючи глобальні мережі обробки трафіку. Водночас їхнім фундаментальним недоліком є неможливість повноцінного контролю локальних каналів витоку інформації, таких як знімні носії або друк. Це змушує організації будувати гібридні схеми захисту, що

автоматично збільшує складність управління політиками безпеки та вимагає використання додаткових брокерів доступу, що часто суперечить методикам оцінки ризиків при побудові комплексних систем [27, 39].

Окрему категорію представляють платформи аналізу внутрішніх загроз, прикладом яких є Proofpoint Insider Threat Management. На відміну від класичних DLP, ці системи фокусуються на візуалізації контексту дій користувача, що дозволяє офіцерам безпеки буквально бачити послідовність маніпуляцій з файлами перед їх викраденням. Перевагою таких рішень є висока швидкість проведення розслідувань та здатність ідентифікувати складні сценарії витоків, які маскуються під легітимні операції. Проте широке впровадження таких інструментів обмежується етичними та юридичними аспектами моніторингу, оскільки вони збирають значну кількість метаданих про діяльність персоналу, що може вступати у протиріччя з вимогами GDPR або положеннями EU AI Act щодо захисту прав працівників. Також недоліком є висока вартість ліцензування, яка часто перевищує бюджет традиційних засобів захисту [36, 32].

Аналіз вітчизняного ринку та нормативної бази, зокрема вимог НД ТЗІ 2.5-004-24, виявляє ще одну суттєву проблему існуючих закордонних рішень - відсутність підтримки національних стандартів криптографічного захисту та специфічних лінгвістичних особливостей. Більшість глобальних вендорів не забезпечують належної якості морфологічного аналізу української мови, що дозволяє зловмисникам обходити захист за допомогою простих маніпуляцій із текстом. Це створює нагальну потребу в розробці систем, які б поєднували потужність світових технологічних досягнень із гнучкістю локалізації під національні потреби та стандарти, що описується в останніх публікаціях щодо менеджменту кібербезпеки [34, 37].

Крім того, розвиток технологій генеративного штучного інтелекту створив новий виклик для всіх без винятку існуючих рішень. Традиційні DLP-системи виявляються малоефективними при передачі конфіденційних даних у вигляді запитів до великих мовних моделей, оскільки інформація у таких випадках часто

не містить чітких сигнатур або маркерів. Новітні дослідження вказують на те, що навіть лідери ринку демонструють високий рівень пропуску таких витоків, що вимагає впровадження методів семантичної фільтрації в реальному часі. Це підтверджує висновок про те, що сучасні рішення, незважаючи на їхню технологічну досконалість, все ще мають суттєві недоліки в частині виявлення інтелектуальних методів викрадення інформації, що робить задачу розробки більш ефективної системи надзвичайно актуальною [30, 24].

На завершення порівняльного аналізу слід підкреслити, що у 2026 році спостерігається тенденція до конвергенції всіх згаданих підходів у єдині платформи безпеки даних. Найбільш збалансованим вважається підхід, який поєднує точність детермінованого аналізу контенту з гнучкістю поведінкових моделей та хмарною масштабованістю. Проте висока вартість та складність впровадження таких інтегрованих платформ залишає відкритою проблему пошуку оптимальних засобів захисту для організацій, що обробляють конфіденційну інформацію в умовах обмежених ресурсів. Виявлені недоліки існуючих систем, такі як фрагментарність захисту, складність адміністрування та вразливість перед новими методами обходу на основі ШІ, формують технічне підґрунтя для розробки власного програмного рішення, яке б враховувало ці виклики та забезпечувало надійний рівень безпеки [35, 33, 21].

1.3 Підходи до вирішення задачі за темою дослідження

Реалізація високоефективної комп’ютерної системи захисту конфіденційної інформації від витоків вимагає комплексного вибору технологічного стека, який здатен забезпечити баланс між глибиною інспекції даних та продуктивністю кінцевих точок. Основним підходом при проектуванні таких систем є використання багаторівневої архітектури, де збір первинної інформації відбувається на низькому рівні операційної системи, а інтелектуальна обробка та прийняття рішень виноситься на рівень прикладних сервісів або

централізованих аналітичних платформ. Це дозволяє системі функціонувати прозоро для користувача, мінімізуючи затримки при виконанні легітимних операцій.

Для забезпечення надійного перехоплення файлових операцій та мережевої активності без порушення стабільності роботи ядра найбільш доцільним є використання мов програмування з низькорівневим доступом до пам’яті, зокрема C++ та Rust. Використання C++ дозволяє розробляти високопродуктивні драйвери фільтрації файлової системи на базі архітектури FltMgr/Minifilter у Windows, які здійснюють моніторинг операцій введення-виведення на рівні запитів до дискової підсистеми. Водночас вибір Rust у сучасних розробках 2026 року обумовлений його моделлю володіння пам’яттю, яка на етапі компіляції унеможливлює вразливості типу витік пам’яті або використання після звільнення. Це критично важливо для стабільності компонентів, що працюють у режимі ядра, де будь-яка критична помилка призводить до відмови всієї системи.

Особливе місце в сучасних підходах займає технологія eBPF для середовищ на базі Linux. Вона дозволяє динамічно завантажувати програмні модулі в ядро операційної системи для відстеження системних викликів та мережевих подій без необхідності перекомпіляції самого ядра. Такий підхід забезпечує гнучкість при оновленні політик безпеки на серверних потужностях. При перехопленні даних на рівні агентів виникає завдання ефективної міжпроцесної взаємодії. Використання технологій спільної пам’яті або іменованих каналів дозволяє передавати метадані файлів з драйвера до прикладного сервісу для аналізу без суттєвого впливу на швидкість обробки черги системних викликів.

Важливим технологічним підходом є впровадження методів глибокої інспекції пакетів для аналізу мережевого трафіку. Для реалізації високопродуктивних мережевих шлюзів, здатних обробляти значні масиви інформації в реальному часі, часто застосовується мова програмування Go. Її

вбудовані механізми легкоатлетичних потоків та оптимізовані бібліотеки для роботи з мережевими протоколами дозволяють створювати масштабовані проксі-сервіси, що виконують дешифрування SSL/TLS з'єднань через механізми підміни сертифікатів. Це дає змогу системі проводити інспекцію вмісту зашифрованих пакетів у корпоративних месенджерах, веб-пошті та хмарних додатках, виявляючи приховані спроби витоку конфіденційної інформації через захищені канали зв'язку.

Інтелектуальна складова системи, що відповідає за класифікацію неструктурованого контенту, базується на використанні методів машинного навчання та обробки природної мови. Сучасний підхід передбачає застосування архітектури трансформерів, реалізованих за допомогою мови Python та бібліотек PyTorch або TensorFlow. Використання векторних представлень слів дозволяє системі оцінювати семантичну близькість переданої інформації до еталонних зразків конфіденційних документів. Такий підхід вирішує проблему обходу захисту шляхом перефразування або зміни формату файлу, оскільки аналіз фокусується на суті повідомлення, а не на сигнатурах. Для обробки графічних даних, таких як скриншоти робочого столу або скани документів, застосовуються технології комп'ютерного зору на базі нейронних мереж, які здійснюють оптичне розпізнавання символів безпосередньо в конвеєрі обробки даних перед їх відправкою в мережу.

Для побудови централізованого сховища подій та аналітики великих даних використовується стек технологій Big Data. Бази даних із колонковим зберіганням, такі як ClickHouse, забезпечують миттєвий доступ до журналів аудиту, що дозволяє виконувати складні запити за лічені секунди навіть при наявності мільярдів записів. Це критично важливо для проведення ретроспективного аналізу та візуалізації життєвого циклу даних, що дає можливість простежити весь шлях конфіденційного файлу від моменту його створення до спроби несанкціонованої передачі. Взаємодія між розподіленими компонентами системи - хостовими агентами, мережевими шлюзами та сервером

управління - організовується через відмовостійкі брокери повідомлень, наприклад Apache Kafka, що гарантує цілісність інформації про інциденти навіть в умовах нестабільного мережевого з'єднання або тимчасової недоступності центрального вузла.

Контейнеризація за допомогою Docker та оркестрація через Kubernetes дозволяють забезпечити високу доступність серверної частини системи та спрощують процес її оверлейного розгортання в існуючих IT-інфраструктурах. Такий підхід забезпечує автоматичне масштабування потужностей системи аналізу контенту при пікових навантаженнях. Таким чином, синергія низькорівневих системних інструментів на базі C++ та Rust, високопродуктивних мережевих стеків на Go та інтелектуальних моделей аналізу контенту на Python формує технологічний фундамент для створення системи, здатної ефективно протидіяти складним комбінованим методам витоку конфіденційної інформації в умовах сучасних кіберзагроз.

1.4 Постановка задачі

Завданнями роботи є:

1) дослідити процедури функціонування сучасних систем запобігання витокам інформації та канали розповсюдження даних;

2) провести теоретичний аналіз сфери захисту конфіденційної інформації в умовах цифровізації та використання штучного інтелекту;

3) охарактеризувати структуру предметної області та розробити базову модель загроз витоку даних;

4) описати уже існуючі механізми реалізації засобів захисту, виділити наявні проблеми в галузі та шляхи їх вирішення;

5) на основі проведених досліджень визначити основні функції проектованої системи, сформувати низку функціональних та нефункціональних вимог, розробити модель функцій, які система повинна виконувати;

6) підвести підсумки про необхідність розробки інтелектуальної системи захисту на основі контекстного аналізу;

7) сформувати об’єкт та мету для наступних етапів проектування та практичної реалізації;

8) оцінити ступінь виконання поставлених завдань та відповідність отриманих результатів вимогам безпеки.

На основі проведеного аналізу необхідно розробити працездатну підсистему моніторингу та перехоплення конфіденційних даних у комп’ютерній системі захисту від витоків та зробити обґрунтовані висновки на основі виконаної роботи. Даний перелік завдань дозволить системно підійти до проектування архітектури та забезпечити ефективну інтеграцію засобів захисту у робочі процеси організації.

1.5 Висновок до першого розділу

У межах першого розділу проведено аналіз предметної області захисту конфіденційної інформації та виявлено наявні проблеми й завдання в умовах розмиття корпоративного мережевого периметра і неконтрольованого використання штучного інтелекту. Здійснено порівняльний аналіз переваг та недоліків існуючих рішень, що дозволило виявити їхні архітектурні обмеження та відсутність належної лінгвістичної адаптації під українську мову. На основі дослідження технологічних підходів здійснено постановку задачі для наступних етапів проєктування, у якій сформовано об’єкт, мету та перелік вимог до розробки ефективної підсистеми моніторингу й перехоплення конфіденційних даних.

2 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА РОЗПОДІЛЕНОЇ СИСТЕМИ ЗАХИСТУ ІНФОРМАЦІЇ ВІД ВИТОКІВ

2.1 Загальна концептуальна модель та архітектура системи

Сучасні інформаційні інфраструктури організацій характеризуються високим ступенем гетерогенності, децентралізації та постійним обміном даними як всередині корпоративного периметра, так і за його межами. Історично перші системи запобігання витокам інформації будувалися за монолітною або виключно шлюзовою архітектурою. У таких системах весь аналіз даних відбувався на центральному сервері або на мережевому екрані, що створювало єдину точку відмови та призводило до суттєвих затримок у маршрутизації корпоративного трафіку формуючи так звані пляшкові горлечока [5,6].

З розвитком технологій, переходом до гібридних форматів роботи та масовим впровадженням хмарних обчислень, класичний мережевий периметр втратив свої чіткі межі. Робоча станція співробітника або кінцева точка тепер часто знаходиться поза межами фізичного контролю організації, підключаючись до корпоративних ресурсів через незахищені мережі. У таких умовах побудова ефективної комп'ютерної системи захисту від витоків інформації вимагає переходу від класичних рішень до гнучкої розподіленої архітектури, побудованої з урахуванням концепції нульової довіри [20, 24].

Розроблена комп'ютерна система захисту від витоків інформації є комплексним прикладом розподіленої універсальної інформаційно-аналітичної системи. Її архітектурна концепція базується на принципах децентралізації процесів обробки даних, де компоненти функціонують відносно автономно, проте підпорядковуються єдиним криптографічно та логічно захищеним політикам безпеки, постійно синхронізуючи результати своєї роботи з координаційним центром.

Концептуально архітектуру розробленої системи було декомпозовано на три основні логічні та фізичні макропідсистеми (рис. 2.1):

– підсистема централізованого управління та агрегації даних;

– підсистема периферійного моніторингу кінцевих точок;

– підсистема мережевого інспектування та глибокого аналізу трафіку.

Такий трирівневий розподіл забезпечує ешелонований захист. Навіть якщо зловмисник або інсайдер знайде спосіб обійти один із рівнів контролю (наприклад, шляхом спроби відключення локального агента), мережевий рівень перехопить спробу передачі конфіденційних даних у зовнішнє середовище.

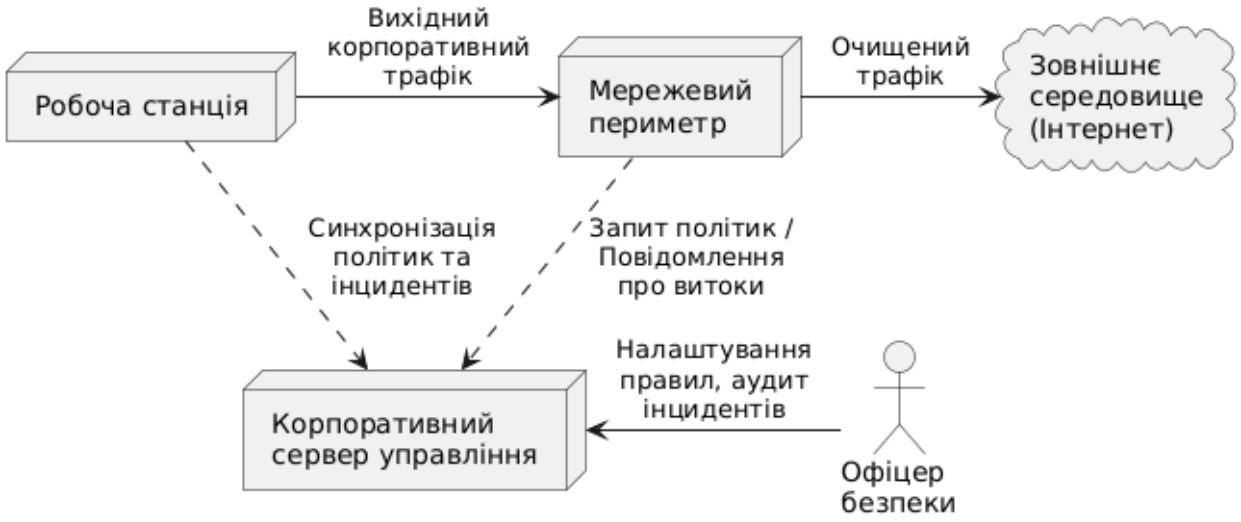


Рисунок 2.1 - Загальна архітектура DLP системи

Центральним ядром, що забезпечує узгодженість функціонування всього комплексу, є підсистема управління та агрегації. З точки зору програмної архітектури, вона реалізована у вигляді мікросервісного вебдодатка на базі мови Python та фреймворку Flask. Вибір Flask як архітектурного фундаменту для серверної частини обумовлений його легковаговістю, високою швидкістю маршрутизації запитів та відповідністю стандарту WSGI, що дозволяє у майбутньому легко інтегрувати додаток із промисловими вебсерверами, такими як Nginx або Apache, для балансування навантаження.

Серверна частина виконує роль координаційного центру та єдиного джерела істини для всієї інфраструктури безпеки.

Архітектурно сервер розділений на два основні блоки: модуль надання API-інтерфейсів для програмних агентів та модуль вебінтерфейсу для візуалізації даних офіцеру з кібербезпеки.

Важливим компонентом цієї підсистеми є шар зберігання даних . У системі використано класичну реляційную модель бази даних на базі СУБД MySQL. Вибір реляційної архітектури забезпечує дотримання принципів атомарності, узгодженості, ізольованості, довговічності, що є критично важливим критерієм для систем інформаційної безпеки, де втрата журналу аудиту є недопустимою.

База даних містить дві ключові сутності, зв'язані логікою роботи системи. Першою з них є таблиця політик безпеки, яка зберігає правила фільтрації. Дані в цій таблиці типізовані, зокрема як заборонені слова, регулярні вирази та критичні розширення файлів, що дозволяє клієнтським агентам формувати оптимізовані структури в оперативній пам'яті під час перевірок. Другою сутністю виступає таблиця інцидентів, яка слугує центральним журналом аудиту. Архітектура цієї таблиці передбачає фіксацію часових міток, ідентифікаторів користувачів, векторів атаки та відповідних типів порушень, а також деталей щодо вжитих превентивних заходів.

Підсистема моніторингу кінцевих точок архітектурно втілює сучасну парадигму периферійних обчислень. У класичних системах моніторингу агенти часто виконують роль тонких клієнтів, завдання яких зводиться лише до збору сирих даних, натискань клавіш, дампу мережевого трафіку, та відправки їх на сервер для подальшого аналізу. Такий підхід має суттєвий архітектурний недолік: він експоненційно збільшує навантаження на пропускну здатність корпоративної мережі та створює затримки, протягом яких конфіденційні дані можуть бути безповоротно втрачені [5, 11].

У розробленій системі агент є товстим клієнтом, наділеним власними аналітичними потужностями. Замість трансляції дій користувача на центральний сервер, агент самостійно завантажує актуальні політики безпеки, зберігає їх у власному локальному кеші оперативної пам'яті і виконує весь семантичний та

сигнатурний аналіз контенту безпосередньо на процесорі кінцевої точки. Це значно оптимізує час відгуку системи на загрозу зменшуючи його до мілісекунд.

Внутрішня архітектура агента побудована на принципах жорсткої асинхронності та багатопотоковості. Замість єдиного монолітного процесу, архітектура агента передбачає ініціалізацію пулу ізольованих фонових потоків, кожен з яких відповідає за контроль конкретного апаратного або програмного каналу потенційного витоку даних. Така модульна структура дозволяє досягти високої стабільності: помилка або зависання в одному модулі, наприклад під час читання пошкодженого USB-носія, не призводить до зупинки інших модулів захисту.

Для забезпечення можливості втручання в системні процеси операційної системи, агент активно використовує низькорівневі системні виклики та механізми перехоплення через спеціалізовані бібліотеки взаємодії з ядром ОС.

Архітектура агента включає такі ключові модулі (рис 2.2):

1) Модуль контролю оперативної пам'яті та буфера обміну;
2) Модуль глибокої інспекції змінних носіїв;
3) Модуль контекстно-залежного блокування;
4) Модуль перехоплення клавіатури;
5) Модуль контролю запису на оптичні носії.

Кожен із зазначених компонентів вирішує специфічні завдання в межах загальної системи захисту та має власні архітектурні особливості.

Першим розглянемо модуль контролю оперативної пам'яті та буфера обміну. Архітектурно він реалізований як постійний слухач стану, що порівнює поточний вміст буфера з попереднім хешем і здійснює евристичний аналіз лише тоді, коли виявляє нові дані.

Розглянемо модуль глибокої інспекції змінних носіїв, який базується на принципах фонового обходу файлової системи. Цей модуль формує граф відомих файлів і відстежує дельти, що дозволяє суттєво економити ресурси процесора і не перевіряти одні й ті самі файли двічі. Якщо ж фіксується

порушення, система запускає алгоритм безпечного затирання даних, який перезаписує сектори диска псевдовипадковими даними ще до виклику системної команди видалення.

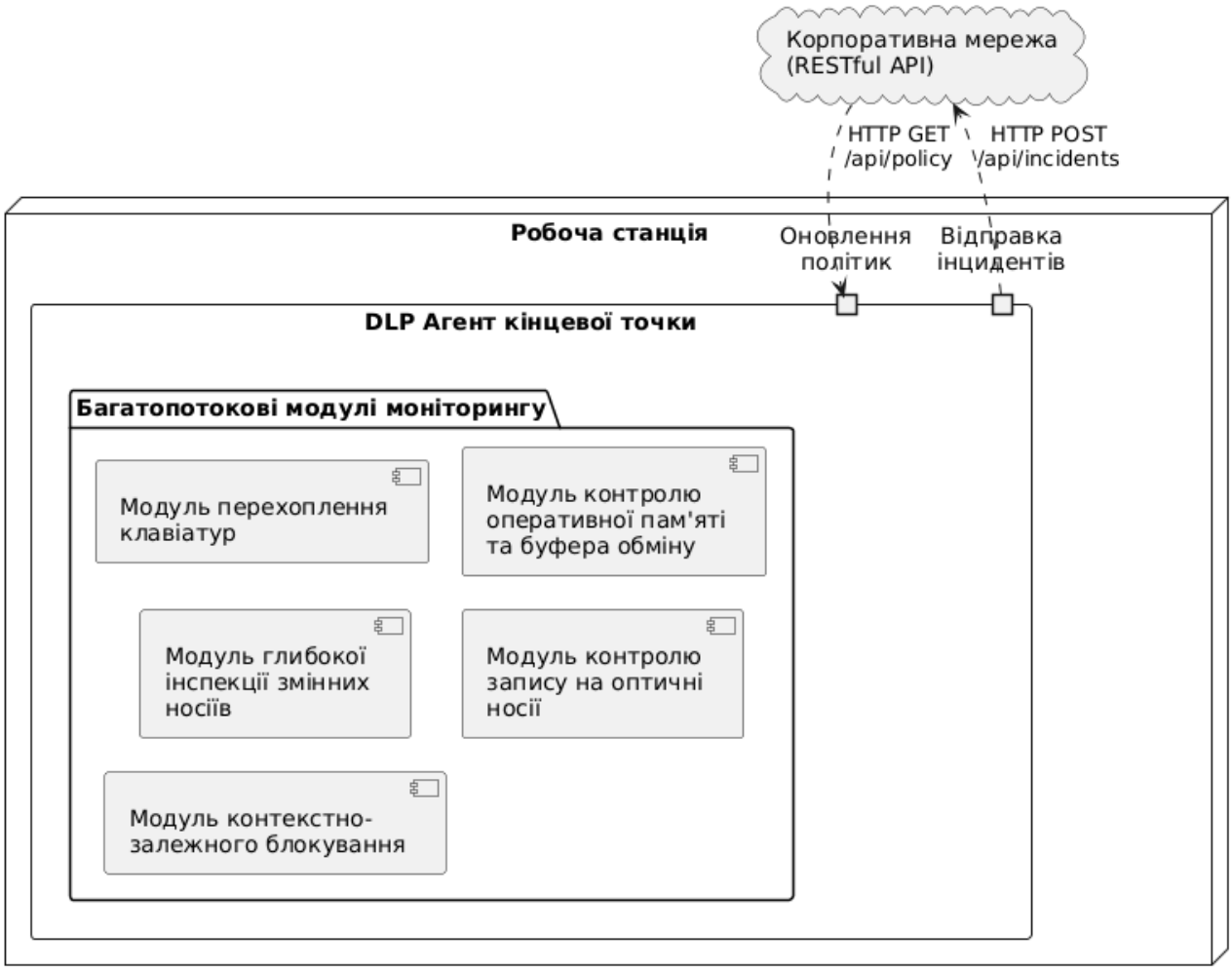


Рисунок 2.2 - Архітектура кінцевих точок

Перейдемо до модуля контекстно-залежного блокування. Він використовує архітектуру на основі подій і постійно зчитує дескриптори та заголовки активних вікон. У разі виявлення конфіденційного контексту модуль виконує сканування таблиці процесів та відправляє системний сигнал KILL тим процесам, які мають технічну можливість зробити знімок екрана.

Наступним компонентом виступає модуль перехоплення клавіатури, що архітектурно вбудовується безпосередньо в ланцюг обробки переривань

клавіатури на рівні операційної системи. Він формує кільцевий буфер заданого розміру, на якому в режимі реального часу виконуються операції пошуку за шаблоном на основі зкомпільованих регулярних виразів.

Також важливим є модуль контролю запису на оптичні носії, який функціонує як спеціалізований монітор директорій кешування. Оскільки сучасні операційні системи тимчасово копіюють файли у приховані системні папки перед безпосереднім записом на диск, цей модуль безперервно відстежує операції вводу та виводу саме в цих цільових локаціях. Якщо до буфера підготовки потрапляє конфіденційний файл, модуль миттєво перехоплює транзакцію, безпечно затирає кешований файл та скасовує системний виклик запису, що повністю унеможливлює витік інформації на фізичний носій.

Підсистема мережевого інспектування виступає третім, ешелонованим рубежем захисту. Архітектурно вона реалізована як прозорий проксі-сервер, розгорнутий на межі корпоративної мережі (рис. 2.3). Головним архітектурним викликом для цієї підсистеми є те, що понад 90% сучасного вебтрафіку зашифровано за протоколами TLS/SSL.

Для вирішення цієї проблеми мережевий перехоплювач використовує архітектурний патерн людина посередині у легітимних цілях. Архітектура передбачає генерацію на льоту підроблених, але довірених у межах корпоративної мережі сертифікатів безпеки. Коли користувач намагається відправити дані на зовнішній сервер, шлюз перехоплює з'єднання, встановлює захищений канал з клієнтом від імені цільового сервера, розшифровує пакет, проводить глибоку інспекцію, після чого встановлює друге захищене з'єднання з реальним зовнішнім сервером і передає очищений трафік.

Усередині мережевого вузла застосовано архітектуру обробки даних у пам'яті. Замість того, щоб зберігати перехоплені файли на жорсткий диск проксі-сервера, що різко знизило б пропускну здатність мережі, підсистема створює байтові потоки в оперативній пам'яті, звідки вони безпосередньо зчитуються парсерами документів, наприклад, модулями для розбору форматів .docx або

.pdf. Така архітектура забезпечує мінімальну затримку при інспекції великих обсягів мережевих даних.

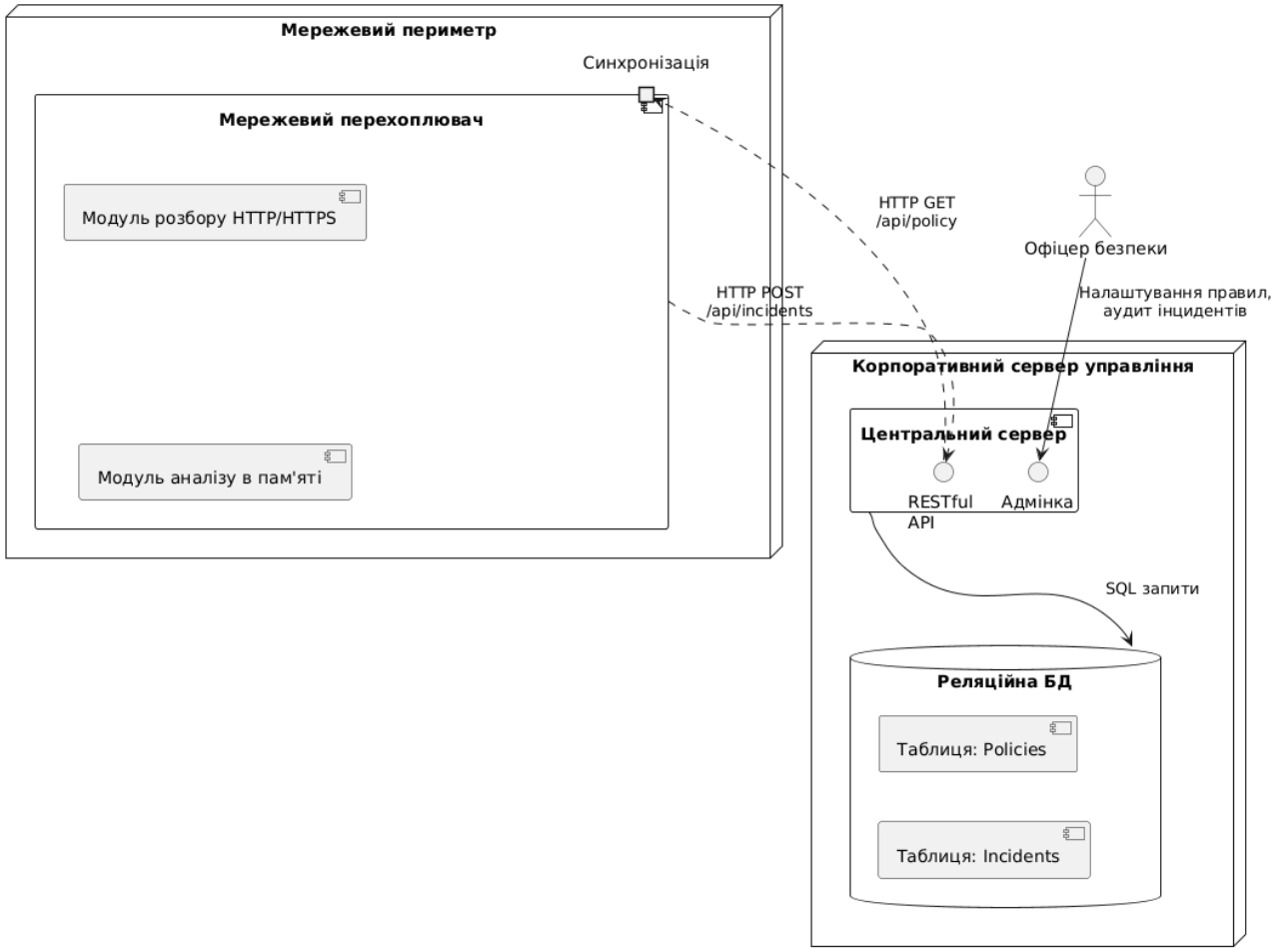


Рисунок 2.3 - Архітектура серверної частини та мережевого перехоплювача

Для забезпечення інтероперабельності між усіма компонентами системи, розроблена архітектура використовує принцип слабкої зв'язності. Взаємодія між підсистемами здійснюється виключно через стандартизований програмний інтерфейс REST API. Використання протоколу HTTP з форматом серіалізації даних JSON дозволяє системі бути агностичною до платформ: сервер може бути розгорнутий на ОС Linux, а агенти функціонувати на ОС Windows, без виникнення конфліктів типів даних.

Забезпечення відмовостійкості є фундаментальною вимогою до спроєктованої архітектури. Розподілений характер системи вирішує критичну вразливість централізованих архітектур. Механізм синхронізації побудований за моделлю Pull коли агенти періодично ініціюють запити на отримання нових політик, а не за моделлю Push коли сервер намагається проштовхнути оновлення всім клієнтам. Цей архітектурний шаблон забезпечує стабільність системи: якщо центральний сервер тимчасово стає недоступним внаслідок збою мережі або DDoS-атаки, агенти кінцевих точок та мережеві шлюзи не припиняють свою роботу. Вони продовжують здійснювати контроль та блокувати витоки, спираючись на останню закешовану версію політик безпеки. Всі зареєстровані інциденти також можуть локально буферизуватися та відправлятися пакетно після відновлення зв'язку з координаційним центром.

Щодо масштабованості, архітектура дозволяє застосовувати як вертикальне, так і горизонтальне масштабування. Збільшення кількості користувачів у мережі, встановлення нових агентів, практично не створює додаткового обчислювального навантаження на центральний сервер, оскільки основний тягар розбору контенту лежить на обчислювальних потужностях самих робочих станцій. Сервер лише обслуговує короткі легкоатлетичні запити до API, що дозволяє одній серверній ноді ефективно контролювати тисячі кінцевих точок одночасно.

Таким чином, розроблена архітектура розподіленої універсальної системи поєднує в собі централізоване адміністрування, високопродуктивний паралельний аналіз на кінцевих точках та глибоку мережеву інспекцію. Це створює надійне підґрунтя для побудови стійкої кіберфізичної системи моніторингу, що відповідає актуальним вимогам інформаційної безпеки.

2.2 Організація децентралізованої взаємодії компонентів у периферійній інфраструктурі

Специфіка функціонування комп'ютерних систем захисту від витоків інформації на рівні кінцевих точок вимагає забезпечення безперервного, одночасного та незалежного контролю за множиною різнорідних каналів передачі даних. Використання класичної однопотокової моделі виконання програмного коду для таких завдань є фундаментально неможливим з огляду на архітектуру сучасних операційних систем та природу операцій введення-виведення.

В однопотоковій моделі інструкції процесора виконуються суворо послідовно. Якщо програма ініціює операцію введення-виведення - наприклад, сканування великого масиву файлів на підключеному USB-накопичувачі або очікування відповіді від центрального сервера через мережу - весь потік блокується до завершення цієї операції операційною системою. У контексті кібербезпеки це означає, що поки система витрачає кілька секунд на аналіз флеш-накопичувача, зловмисник або інсайдер має можливість безперешкодно скопіювати конфіденційні дані через буфер обміну або відправити їх у хмарне середовище. У цей момент модулі контролю інших каналів перебуватимуть у стані очікування процесорного часу.

Для вирішення цієї проблеми у розробленій системі застосовано парадигму паралельних обчислень та багатопотоковості. Розпаралелювання дозволяє розділити монолітний процес моніторингу на набір ізольованих фонових потоків, кожен з яких конкурує за ресурси центрального процесора незалежно від інших завдяки планувальнику завдань операційної системи.

Теоретичну доцільність розпаралелювання в даній системі можна описати за допомогою закону Амдала, який ілюструє потенційне прискорення виконання програми при розділенні її на паралельні потоки.

Закон виражається формулою 2.1:

$$S(n) = \frac{1}{(1-P)+\frac{P}{n}}, \quad (2.1)$$

де *S(n)* – теоретичне прискорення виконання;

P – частка алгоритму, яка може бути розпаралелена у випадку DLP агента, це практично всі модулі моніторингу, тобто *P* наближається до 0.95;

1-P – частка алгоритму, що виконується послідовно (ініціалізація змінних, запуск потоків);

n – кількість обчислювальних потоків.

Оскільки більшість завдань DLP-агента є залежними від операцій введення-виведення обмеженими швидкістю дискової або мережевої підсистеми, а не процесорною потужністю, використання потоків операційної системи дозволяє досягти високої ефективності. Навіть в умовах використання мов програмування з глобальним блокуванням інтерпретатора, потоки звільняють блокування під час операцій введення-виведення, забезпечуючи справжній паралелізм на рівні системи.

Першочерговим напрямком моніторингу виступає безперервний контроль оперативної пам'яті та буфера обміну (рис. 2.4). Буфер обміну операційної системи є одним із найпоширеніших, найшвидших та найменш контрольованих векторів витоку інформації. Функціональне завдання цього модуля полягає у безперервному зчитуванні текстового вмісту глобального буфера ОС, порівнянні його поточного стану з попереднім, для уникнення дублювання перевірок та семантичному аналізі тексту.

Модуль функціонує на базі нескінченного циклу, періодично опитуючи API операційної системи. Якщо реалізувати цей цикл в основному потоці програми, він повністю монополізує виконання. Винесення цього завдання в окремий паралельний потік дозволяє системі здійснювати поллінг буфера обміну з частотою один раз на секунду, не заважаючи іншим процесам. У разі виявлення регулярних виразів або заборонених слів, наприклад, фінансових даних або грифів таємності, потік ініціює системний виклик для миттєвого очищення буфера пам'яті, генеруючи асинхронну подію порушення політики.

Поряд із програмними каналами, невід'ємною складовою захисту є глибока інспекція та блокування знімних носіїв. Контроль фізичних портів вводу-виводу, зокрема USB-накопичувачів, є критичною вимогою міжнародних стандартів інформаційної безпеки, таких як ISO/IEC 27001. Завдання цього модуля полягає у динамічному виявленні нових знімних дисків, побудові дерева каталогів та виконанні глибокого контентного аналізу файлів, що копіюються на носій.

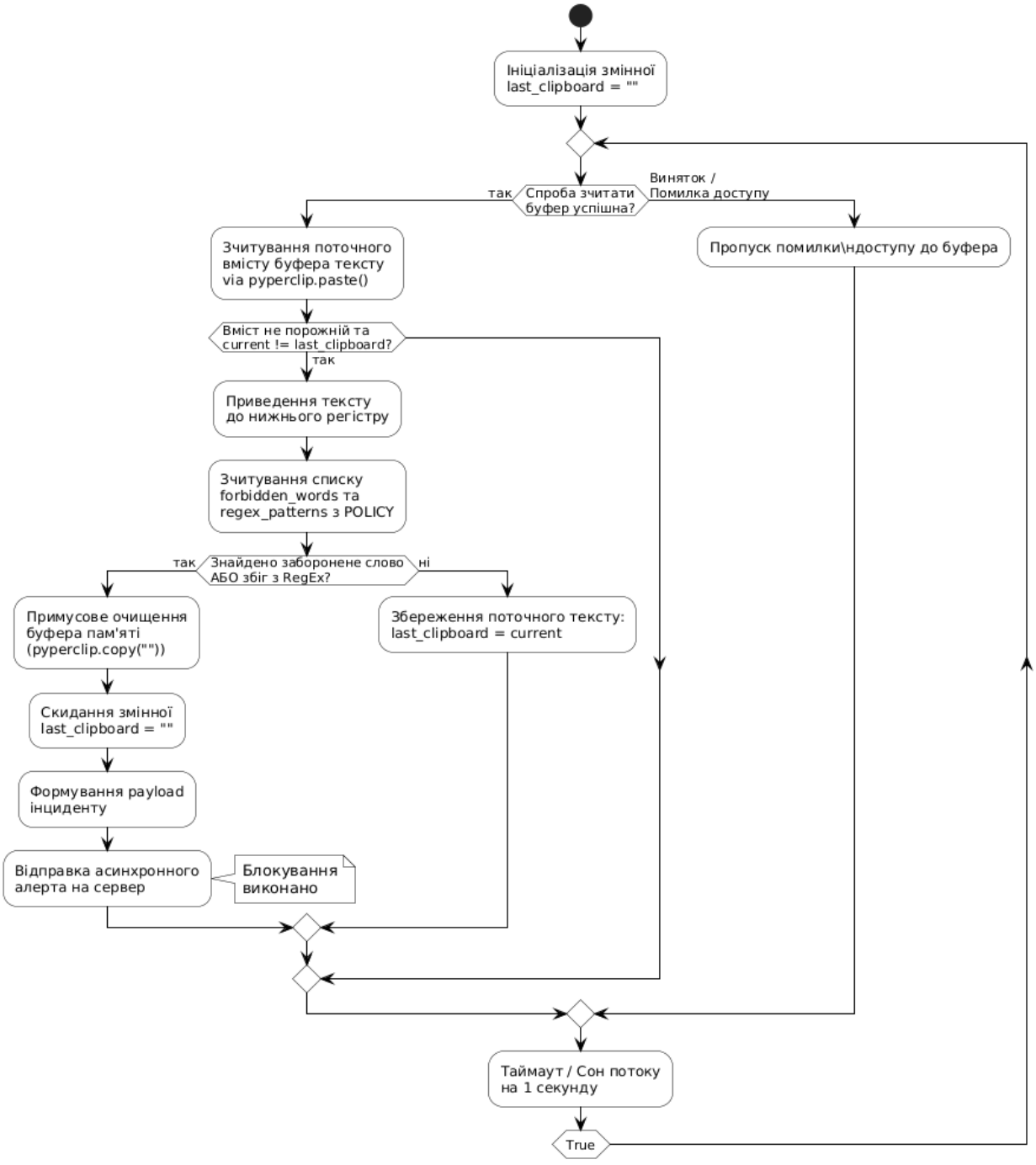


Рисунок 2.4 - Алгоритм моніторингу буфера обміну

Це завдання характеризується найвищою алгоритмічною складністю та латентністю вводу та виводу. Читання структури файлової системи через зовнішню шину та подальший парсинг документів, розпакування архівів .docx, вилучення тексту з .pdf за допомогою модуля PyPDF2, вимагає значного часу. Якщо це завдання не розпаралелити, підключення співробітником флеш-накопичувача з тисячами дрібних файлів призведе до ефекту заморожування всього DLP-агента на кілька хвилин.

У виділеному паралельному потоці цей модуль асинхронно формує локальні множини відомих шляхів до файлів, виявляє новостворені файли, обчислюючи дельту стану та перевіряє їх вміст. У разі виявлення витоку, саме цей потік викликає функцію алгоритмічного затирання secure_wipe, яка перезаписує сектори диска псевдовипадковими байтами перед системним видаленням файлу, що робить його відновлення неможливим.

Специфічним напрямком контролю виступає моніторинг системних черг запису оптичних дисків. Незважаючи на зниження популярності оптичних носіїв, можливість запису CD/DVD залишається вбудованою функцією сучасних ОС і часто ігнорується базовими засобами захисту, створюючи прихований канал витоку даних. Завдання модуля - моніторити системні черги підготовки файлів до запису на оптичні диски.

Як і у випадку з USB, моніторинг локальних каталогів запису вимагає періодичних операцій читання диска. Виділення цього завдання в окремий потік дозволяє агенту незалежно обходити дерево директорій запису, виявляючи файли із забороненими розширеннями, визначеними у глобальній політиці, або файли, що містять конфіденційний текст, блокуючи спробу запису ще до початку роботи лазера оптичного приводу (рис. 2.5).

Наступним критичним елементом захисту є механізм перехоплення системних переривань клавіатури. Сучасні методи компрометації даних включають не лише пряме копіювання файлів, а й ручний набір конфіденційної інформації.

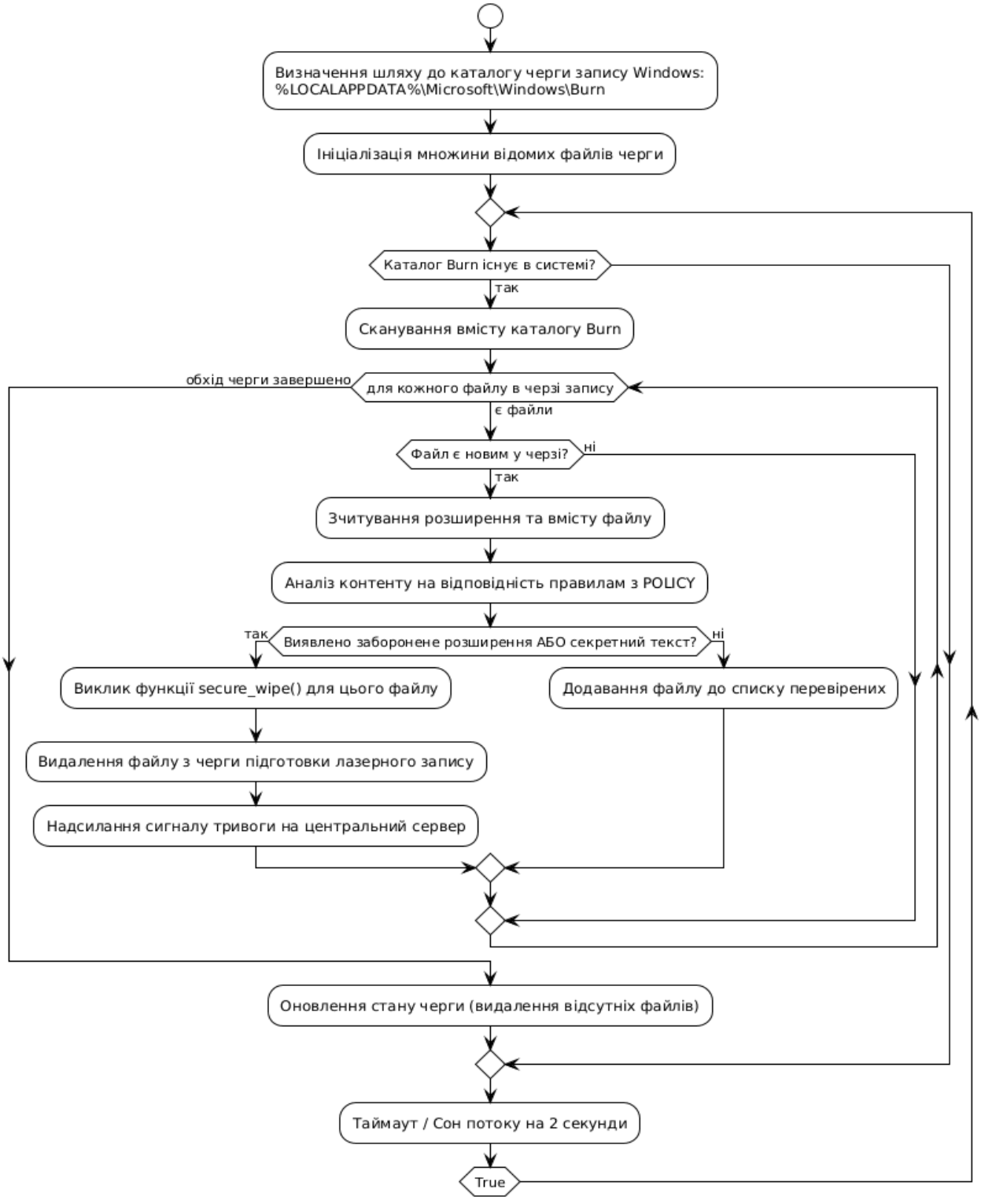


Рисунок 2.5 - Алгоримт роботи потоку контролю запису оптичних дисків

Завдання модуля полягає в перехопленні системних подій натискання клавіш на рівні драйвера клавіатури.

Перехоплення клавіатури є подійно-орієнтованим завданням, яке вимагає жорстко детермінованої реакції у режимі реального часу. Допустима затримка обробки не повинна перевищувати 20-50 мілісекунд. Будь-яка затримка в основному потоці призведе до видимого користувачем залипання клавіатури, що порушить ергономіку роботи та розкриє наявність системи моніторингу.

У розпаралеленому середовищі потік формує динамічний кільцевий буфер вводу, розмір якого обмежено останньою сотнею символів. Аналізатор у фоновому режимі перевіряє буфер на збіги із забороненими шаблонами. Якщо виявлено спробу ручного введення таємниці, потік програмно генерує серію сигналів клавіші Backspace, фізично видаляючи набраний текст з екрана.

Логічним доповненням системи виступає механізм контекстно-залежного блокування скриншотів. Витік інформації через оптичні та графічні канали такі як створення знімків екрана становить серйозну загрозу, оскільки зображення набагато важче піддаються автоматизованому аналізу в реальному часі. Завдання цього компонента - привентивно блокувати спроби запуску програм для створення скриншотів, якщо на екрані відображається конфіденційна інформація.

Даний модуль повинен безперервно звертатися до системних бібліотек, щоб зчитувати дескриптор та заголовок активного вікна. Окрім цього, модуль здійснює повний перебір списку активних процесів операційної системи, що є ресурсомісткою операцією.

Виконання цієї задачі у паралельному потоці з таймером гарантує, що система встигне ідентифікувати запуск забороненого процесу і відправити йому сигнал завершення рівно в той момент, коли користувач працює з таємним документом, не перериваючи роботу мережевих чи дискових сканерів.

Важливою складовою комплексу є асинхронна мережева синхронізація політик безпеки (рис. 2.6). Система захисту не є статичною; вона повинна адаптуватися до змін у корпоративних правилах.

Завдання цього модуля - періодично встановлювати з'єднання з центральним сервером управління через інтерфейс REST API для завантаження актуальних словників заборонених слів та регулярних виразів, а також відправки накопичених логів.

Взаємодія через транспортні мережеві протоколи є найменш детермінованою операцією. Сервер може бути перевантажений, мережевий пакет може загубитися, маршрутизатор може внести затримку, або з'єднання взагалі може бути відсутнім. Якщо агент чекатиме відповіді в основному потоці, весь процес моніторингу кінцевої точки буде заблоковано на час мережевого тайм-ауту. Виділення окремого потоку-демона для синхронізації дозволяє мережевим запитам виконуватись у фоні.

Потік формує запити із заданим інтервалом; у разі успіху він оновлює глобальні політики в пам'яті, а у разі збою - просто переходить у стан сну, не впливаючи на локальну автономну роботу інших захисних модулів.

Важливим інженерним аспектом при розпаралелюванні вищеописаних шести завдань є проблема управління спільно використовуваними ресурсами оперативної пам'яті. У розробленій архітектурі потоки є незалежними, проте всі вони звертаються до глобального словника політик POLICY для отримання списку заборонених слів та регулярних виразів.

Оскільки потік мережевої синхронізації періодично перезаписує цей словник новими даними з сервера, а інші п'ять потоків одночасно читають його для перевірки файлів чи буфера, виникає ризик так званого стану гонитви. Для забезпечення безпеки потоків у теоретичній моделі системи передбачено використання примітивів синхронізації операційної системи, таких як м'ютекси або механізми глобального блокування.

Це гарантує, що контентний аналіз файлу не завершиться критичною помилкою, якщо політики будуть змінені сервером безпосередньо в момент сканування файлу.

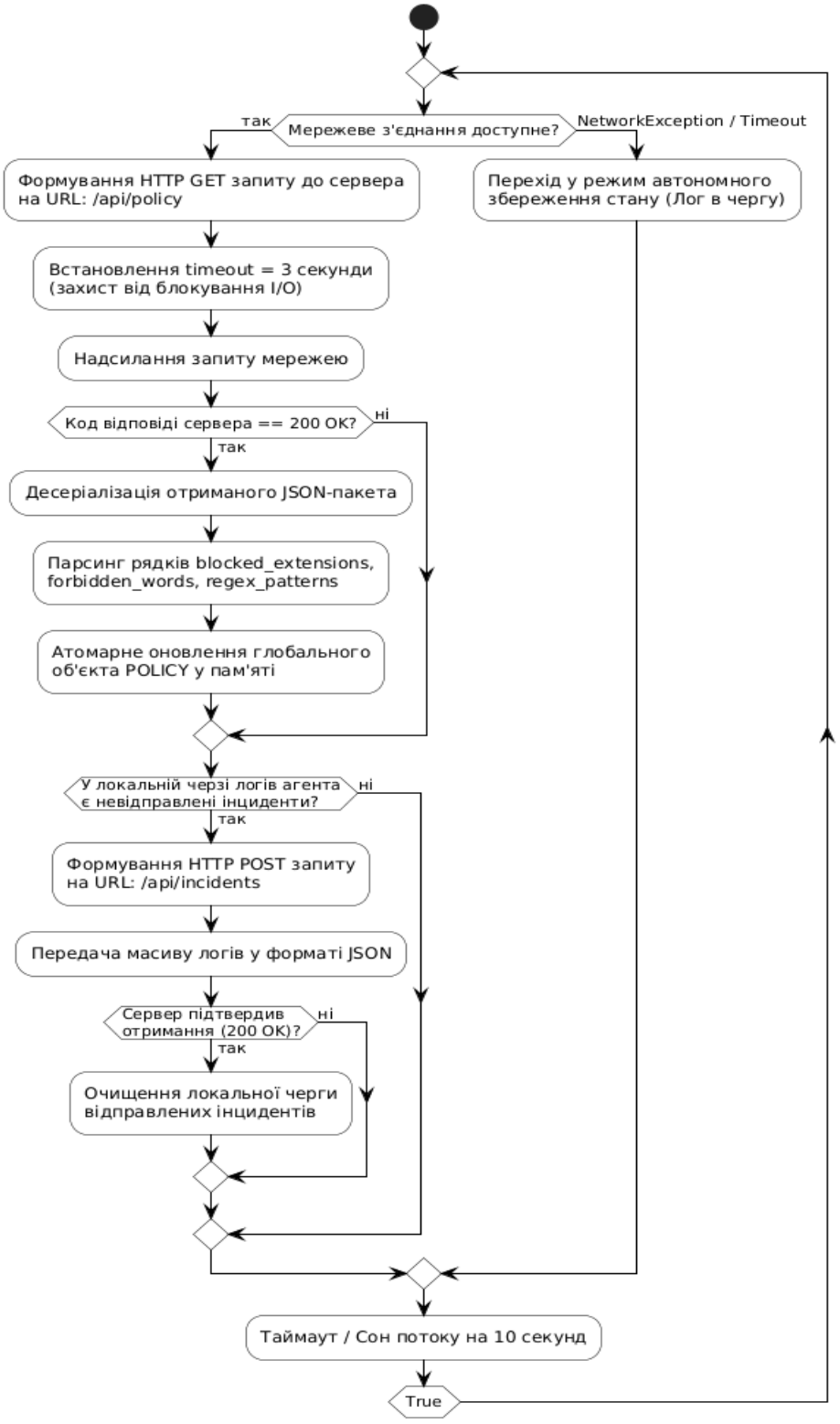


Рисунок 2.6 - Алгоритм роботи мережевої синхронізації

Декомпозиція монолітного процесу моніторингу кінцевої точки на шість ізольованих паралельних завдань є єдино можливим архітектурним рішенням для забезпечення безперервності захисту.

Розпаралелювання дозволяє нівелювати вплив затримок введення-виведення на продуктивність, гарантує миттєву реакцію на події в режимі реального часу особливо при перехопленні клавіатури та буфера обміну та забезпечує повну непомітність роботи системи захисту від витоків для кінцевого користувача. Синергія цих незалежних потоків утворює замкнений контур моніторингу на рівні операційної системи.

2.3 Математичне моделювання паралельних обчислювальних процесів на кінцевих точках

У контексті проєктування сучасних кіберфізичних систем та систем інформаційної безпеки, зокрема DLP-систем, концепція самоорганізації відіграє фундаментальну роль. Традиційні ієрархічні системи управління безпекою, такі як модель Master-Slave, де всі аналітичні рішення ухвалюються виключно центральним вузлом, виявляють свою критичну неефективність в умовах динамічних мережевих середовищ, великої кількості кінцевих точок та безперервного зростання обсягів трафіку.

Теорія складних систем визначає самоорганізацію як здатність системи самостійно, без прямого зовнішнього керування, адаптувати свою внутрішню структуру, алгоритми обробки та поведінку у відповідь на зміни навколишнього середовища або внутрішні апаратні чи мережеві збої. Для розробленої розподіленої універсальної системи запобігання витокам інформації самоорганізація реалізується через три ключові механізми, які програмно закладені в архітектуру:

1) автономне виконання функціональних завдань без залучення адміністратора;

2) динамічну перебудову логічної архітектури;

3) здатність до переміщення обчислювального центру прийняття рішень між компонентами системи.

Основною властивістю розробленої системи є делегування повноважень щодо прийняття рішень на рівень периферійних вузлів. У класичних системах виявлення загроз периферійний агент, зафіксувавши підозрілу активність, генерував запит до центрального сервера і очікував вердикту від адміністратора або центральної бази сигнатур. Така модель створювала неприпустимі затримки, протягом яких конфіденційна інформація могла бути безповоротно скопійована на зовнішній носій.

У запропонованій архітектурі реалізовано парадигму повної функціональної автономності. Центральний сервер та адміністратор виступають виключно у ролі законодавчого органу, який формує глобальні політики безпеки, набори заборонених слів, скомпільовані регулярні вирази, списки критичних розширень файлів. Периферійні агенти виконують роль виконавчої влади. Після асинхронного завантаження цих політик через інтерфейс REST API, агент перетворюється на самодостатню локальну експертну систему [13, 25].

Завдяки багатопотоковій архітектурі, агент самостійно ідентифікує загрозу та застосовує превентивні дії без жодного втручання людини:

1) виявивши заборонений файл на USB-накопичувачі, модуль автономно ініціює алгоритм безпечного затирання, перезаписуючи сектори диска псевдовипадковими даними;

2) виявивши конфіденційний контекст в активному вікні, агент самостійно сканує таблицю процесів ОС і примусово завершує процеси створення скриншотів;

3) при спробі ручного введення таємниці, через перехоплення подій клавіатури, агент діє як зворотний кейлогер, генеруючи віртуальні натискання клавіші Backspace для видалення тексту.

Усі ці захисні дії виконуються за мілісекунди, без надсилання мережевого запиту до сервера на підтвердження дії та без виведення діалогових вікон кінцевому користувачу. Адміністратор системи отримує лише структурований звіт про вже нейтралізований інцидент. Такий підхід гарантує, що людський

фактор, наприклад, відсутність офіцера безпеки на робочому місці в неробочий час або мережеві затримки жодним чином не знижують рівень захищеності кінцевої точки.

Найбільш складною теоретичною та інженерною проблемою розподілених інформаційних систем є забезпечення відмовостійкості. Згідно з фундаментальною CAP-теоремою, у разі розриву мережевого з'єднання розподілена система повинна зробити вибір між узгодженістю даних та доступністю сервісу.

Для систем класу DLP пріоритетом є безперервна доступність механізмів захисту. Тому в розробленій системі програмно реалізовано механізм динамічної перебудови архітектури у разі настання нештатних ситуацій, мережевих збоїв, відмови сервера або маршрутизаторів.

У штатному режимі архітектура системи є чітко вираженою зіркою, де центральний сервер управління з реляційною базою даних є координаційним та інформаційним центром. Агенти діють як сателіти, постійно надсилаючи телеметрію та запитуючи оновлення.

Однак, у випадку втрати зв'язку - наприклад, коли співробітник з корпоративним ноутбуком відключається від офісної мережі та продовжує роботу в офлайн-режимі у відрядженні чи вдома - система зазнає миттєвої логічної перебудови. Цей процес супроводжується ізоляцією обчислювальних потоків, за якої потік мережевої синхронізації агента, зіткнувшись із тайм-аутом або помилкою HTTP-з'єднання, не викликає фатальної зупинки програми, а перехоплює виняток і переходить у режим очікування, дозволяючи іншим п'яти модулям захисту безперешкодно продовжувати роботу з процесором та диском.

Водночас відбувається перебудова внутрішніх інформаційних потоків, оскільки припинення відправки інцидентів на сервер змушує агент динамічно реконструювати конвеєр обробки даних, внаслідок чого зареєстровані події починають накопичуватися у локальному буфері оперативної пам'яті.

Крім того, змінюються принципи роботи з кешем: агент переходить з режиму керованого виконання в режим ізольованої самоорганізації, спираючись виключно на останній відомий стан політик, які зберігаються в оперативній пам'яті. Як тільки мережеве середовище стабілізується, архітектура самоорганізується у зворотному напрямку з переходом до парадигми узгодженості в кінцевому рахунку, за якої потоки агента вивантажують накопичені логи на сервер, оновлюють свої політики, а логічна інфраструктура знову набуває централізованого вигляду [15, 17]. Логічним наслідком такої динамічної перебудови є переміщення обчислювального центру тяжіння або центру прийняття рішень системи.

В умовах стабільного з'єднання центром управління є вебсервер. Проте, при деградації мережі, центр управління динамічно розпадається. Кожен локальний DLP-агент на кінцевій точці бере на себе роль мікро-сервера для власної робочої станції. Він самостійно вирішує, які файли блокувати і які процеси знищувати, стаючи локальним центром авторизації дій.

Додатково, у сценаріях, коли центральний сервер управління виходить з ладу, але корпоративна локальна мережа продовжує працювати, архітектурний фокус частково зміщується на Мережевий Інтерсептор. Оскільки шлюз продовжує перехоплювати трафік від пристроїв, використовуючи закешовані правила та власні потоки, він стає тимчасовим центром фільтрації для всієї підмережі, захищаючи ті пристрої, наприклад, смартфони та гостьові ноутбуки, які не мають встановленого автономного агента (рис. 2.7).

Така здатність архітектури гнучко пульсувати - централізуватись при стабільній інфраструктурі та розпадатися на мережу повністю автономних, самоорганізованих вузлів при збоях - гарантує, що система зберігає свої цільові захисні функції незалежно від стану транспортного середовища. Це є ознакою високонадійної кіб ерфізичної системи моніторингу.

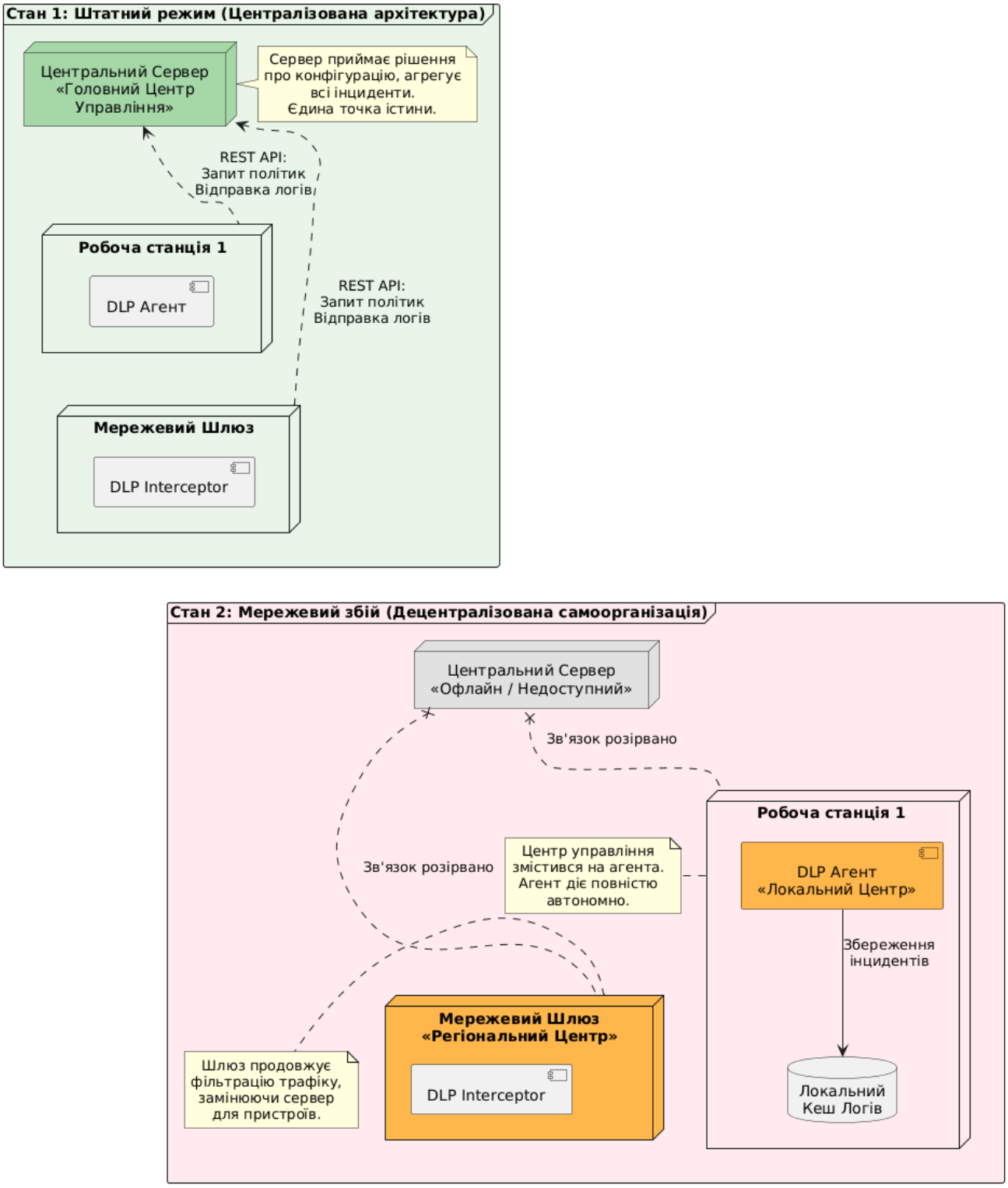


Рисунок 2.7 - Схема організації та перебудови архітектури

2.4 Висновок до другого розділу

У межах другого розділу проведено проєктування децентралізованої трирівневої архітектури комп'ютерної системи захисту та обґрунтовано принципи розподілу обчислювального навантаження в умовах застосування парадигми периферійних обчислень. Здійснено аналіз багатопотокової взаємодії фонових процесів агента на основі закону Амдала, що дозволило визначити механізми їхньої логічної перебудови та автономної самоорганізації в офлайн-режимі згідно з положеннями CAP-теореми. На основі розроблених системних рішень створено теоретичне підґрунтя для наступних етапів дослідження, у якому сформовано архітектурний базис для подальшого проектування та опису низькорівневих алгоритмів функціонування підсистем моніторингу.

3 АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Алгоритми функціонування системи

Ефективність функціонування розподіленої системи захисту від витоків інформації безпосередньо залежить від математичної та логічної коректності алгоритмів, що покладені в основу її компонентів. На етапі програмної реалізації виникає необхідність абстрагування від конкретного синтаксису мови програмування та представлення ключових процесів у вигляді універсального псевдокоду. Це дозволяє проаналізувати часову та просторову складність алгоритмів, перевірити їх на наявність логічних тупиків та забезпечити можливість перенесення архітектурних рішень на будь-які інші програмні платформи.

У межах розробленого програмно-технічного комплексу базову логіку обробки інформації можна розділити на три автономні алгоритмічні контури: алгоритм синхронізації та управління конфігурацією, алгоритми локального контентного моніторингу периферійного агента та алгоритм глибокої інспекції мережевих потоків на рівні шлюзу.

Першим критично важливим алгоритмом є процес запуску DLP-агента, розгалуження його на паралельні обчислювальні потоки та організація регулярного оновлення локального кешу правил безпеки з центрального сервера через REST API.

Особливістю цього алгоритму є превентивне перехоплення мережевих винятків. Якщо центральний вебсервер є недоступним, алгоритм не перериває роботу системи, а застосовує попередню валідну конфігурацію, забезпечуючи автономність кіберфізичної системи (рис. 3.1). Після цього активуються спеціалізовані фонові демони для безперервного моніторингу каналів даних, а цілісність їхнього паралельного доступу до бази політик забезпечується потокобезпечними примітивами синхронізації.

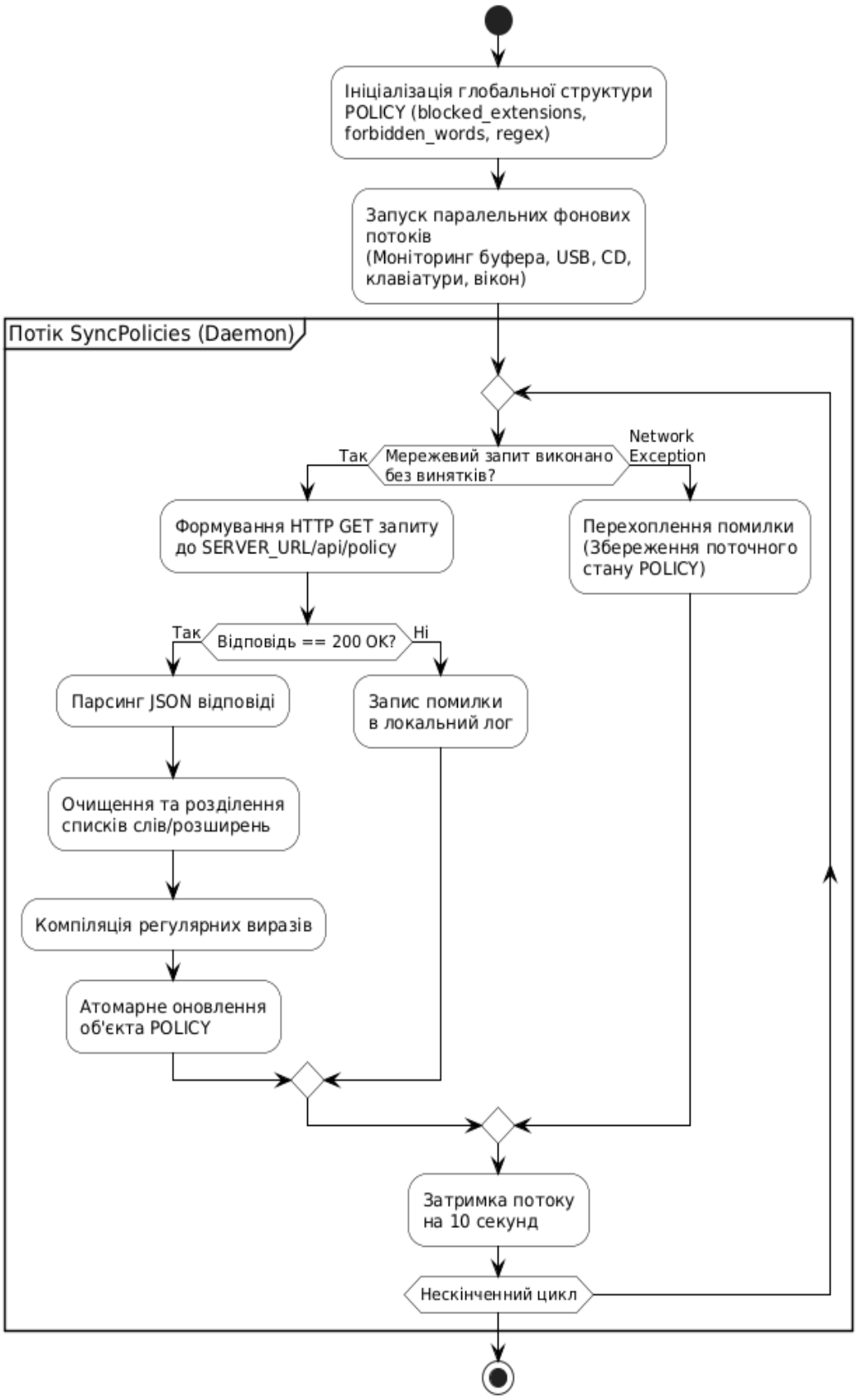


Рисунок 3.1 - Блок-схема алгоритму синхронізації політик безпеки

У разі виявлення локального інциденту, наприклад, спроби запису конфіденційного документа на знімний USB-носій, простого виклику стандартної команди операційної системи для видалення файлу є недостатньо. Стандартні механізми ОС лише очищують покажчик на файл у таблиці MFT або FAT, залишаючи саме тіло файлу, а саме байтовий масив на фізичних секторах диска, що дозволяє зловмиснику легко відновити інформацію за допомогою програмного забезпечення класу інструментів комп'ютерної криміналістики. Для нівелювання цієї загрози розроблено алгоритм гарантованого знищення інформації (рис. 3.2), який здійснює фізичне затирання інформації на рівні абстракції файлової системи перед її остаточним видаленням [18, 19, 21].

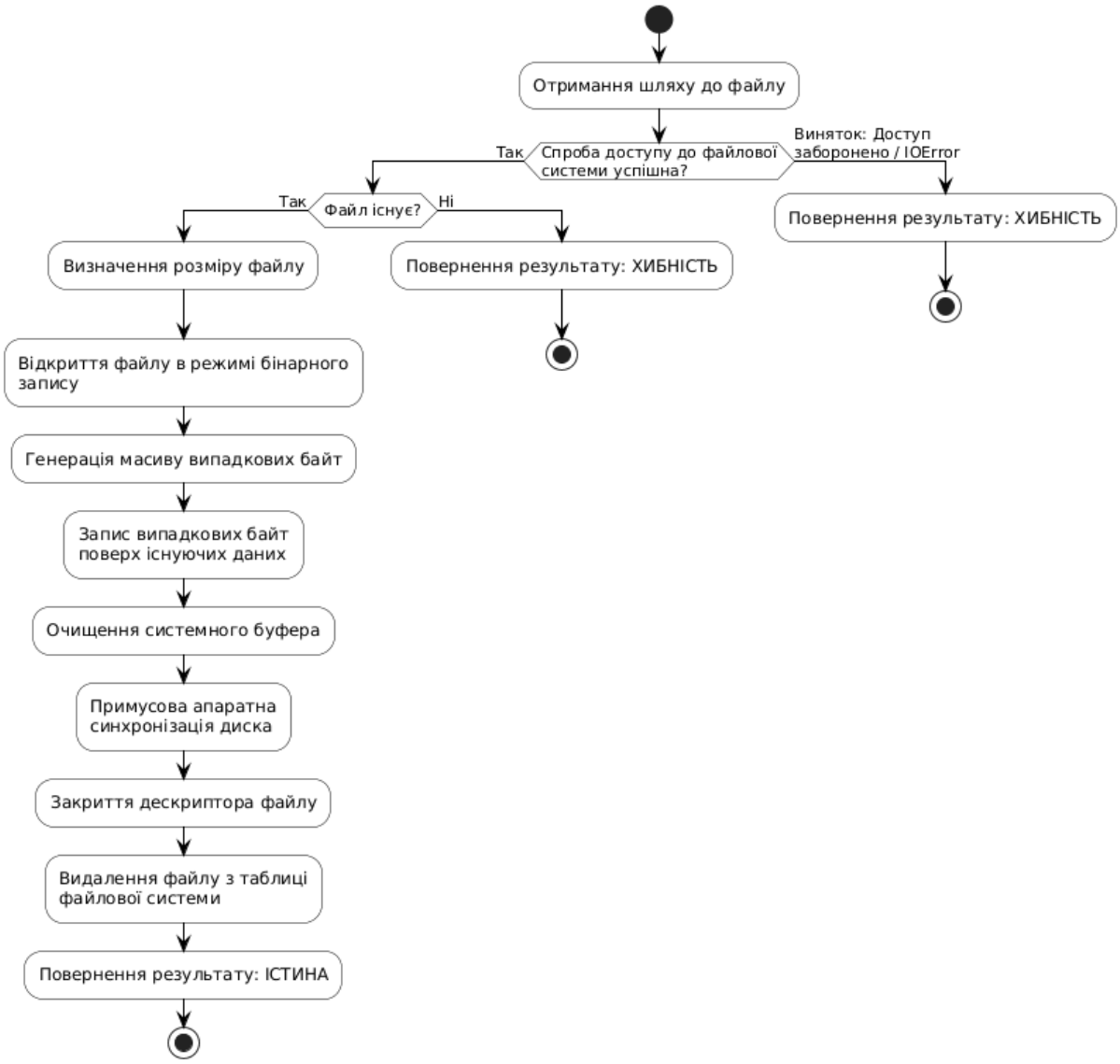


Рисунок 3.2 Блок-схема алгоритму гарантованого знищення інформації

Математична та інженерна доцільність алгоритму полягає в задіянні системного виклику os.fsync. Стандартна функція запису мов високого рівня лише переміщує байтовий масив у внутрішній буфер ядра операційної системи. Якщо в цей момент користувач фізично витягне USB-накопичувач з порту, затирання не відбудеться. Команда fsync примусово блокує потік виконання доти, доки контролер фізичного диска не підтвердить повний запис блоку псевдовипадкових чисел безпосередньо на напівпровідникові чи магнітні комірки носія.

Моніторинг витоків інформації через введення даних з клавіатури вимагає реалізації алгоритму, що функціонує в режимі жорсткого реального часу. Алгоритм базується на концепції низькорівневого перехоплення системних переривань та циклічного аналізу кільцевого буфера символів (рис 3.3).

У разі детектування забороненого патерну, алгоритм здійснює активну компенсаційну дію - програмно надсилає операційній системі коди клавіші Backspace, знищуючи введену інформацію на екрані користувача ще до моменту її потенційного збереження або відправки в месенджері [12, 31].

Останнім критичним вузлом алгоритмічного забезпечення є логіка роботи Мережевого Інтерсептора, що аналізує HTTP/HTTPS трафік на льоту. Головна архітектурна вимога до цього алгоритму - повна відмова від збереження тимчасових файлів на жорсткий диск проксі-шлюзу для виключення додаткових затримок дискової підсистеми та унеможливлення перехоплення цих даних сторонніми процесами. Алгоритм зображений на рисунку 3.4 описує процес перехоплення та аналізу POST-запиту, що містить бінарні дані файлу.

Функція AnalyzeFileInMemory, що викликається в алгоритмі на рисунку 3.4, використовує віртуальні байтові потоки, наприклад, об'єкти класу io.BytesIO.

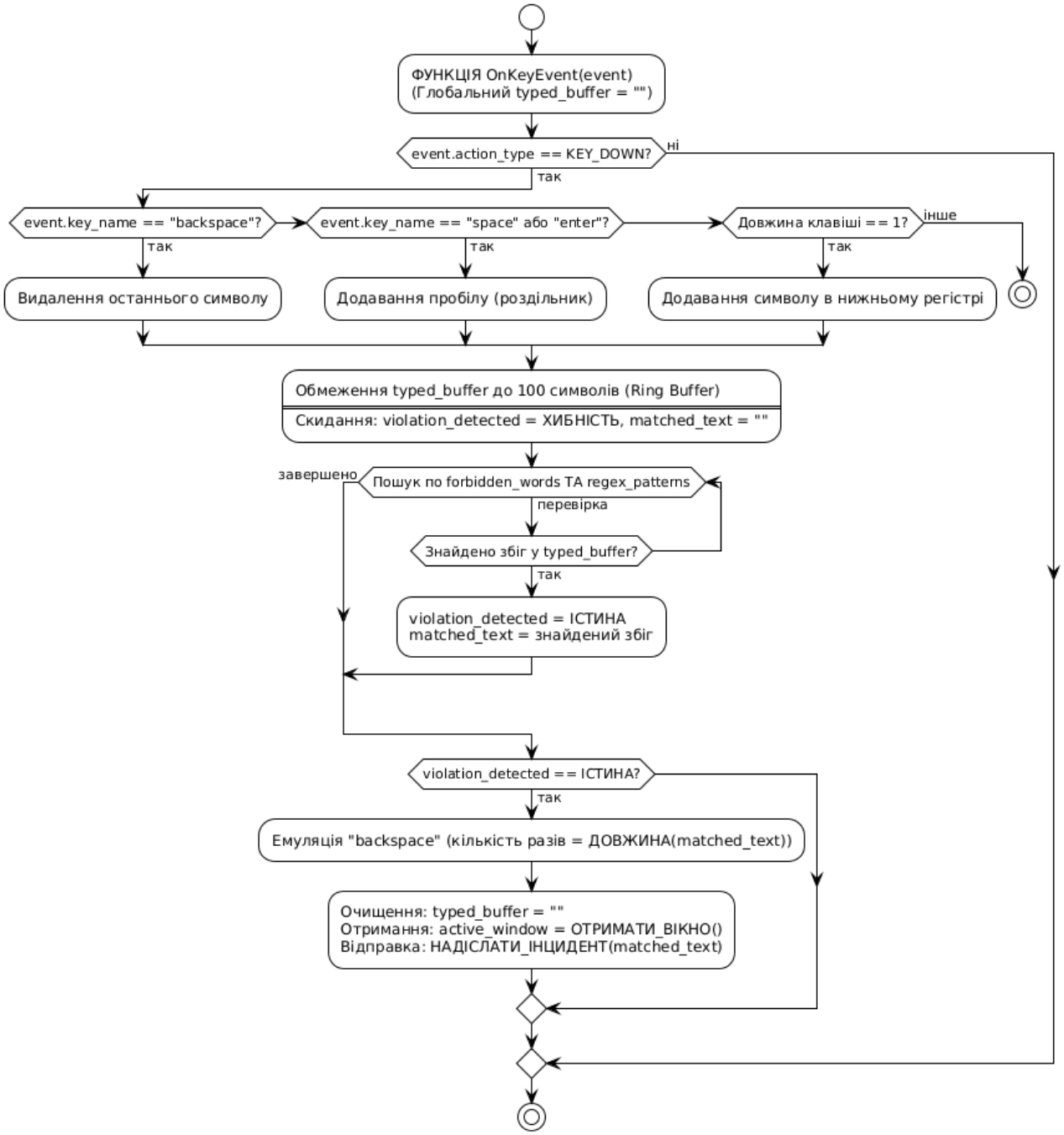


Рисунок 3.3 – Блок-схема алгоритму перехоплення та зворотного стирання тексту

Це дозволяє передавати сирий масив байт file_bytes безпосередньо у програмні модулі розбору офісних документів (docx, pdf), імітуючи для парсерів відкриття звичайного фізичного файлу на диску, що повністю відповідає концепції високопродуктивної обробки інформації в реальному часі.

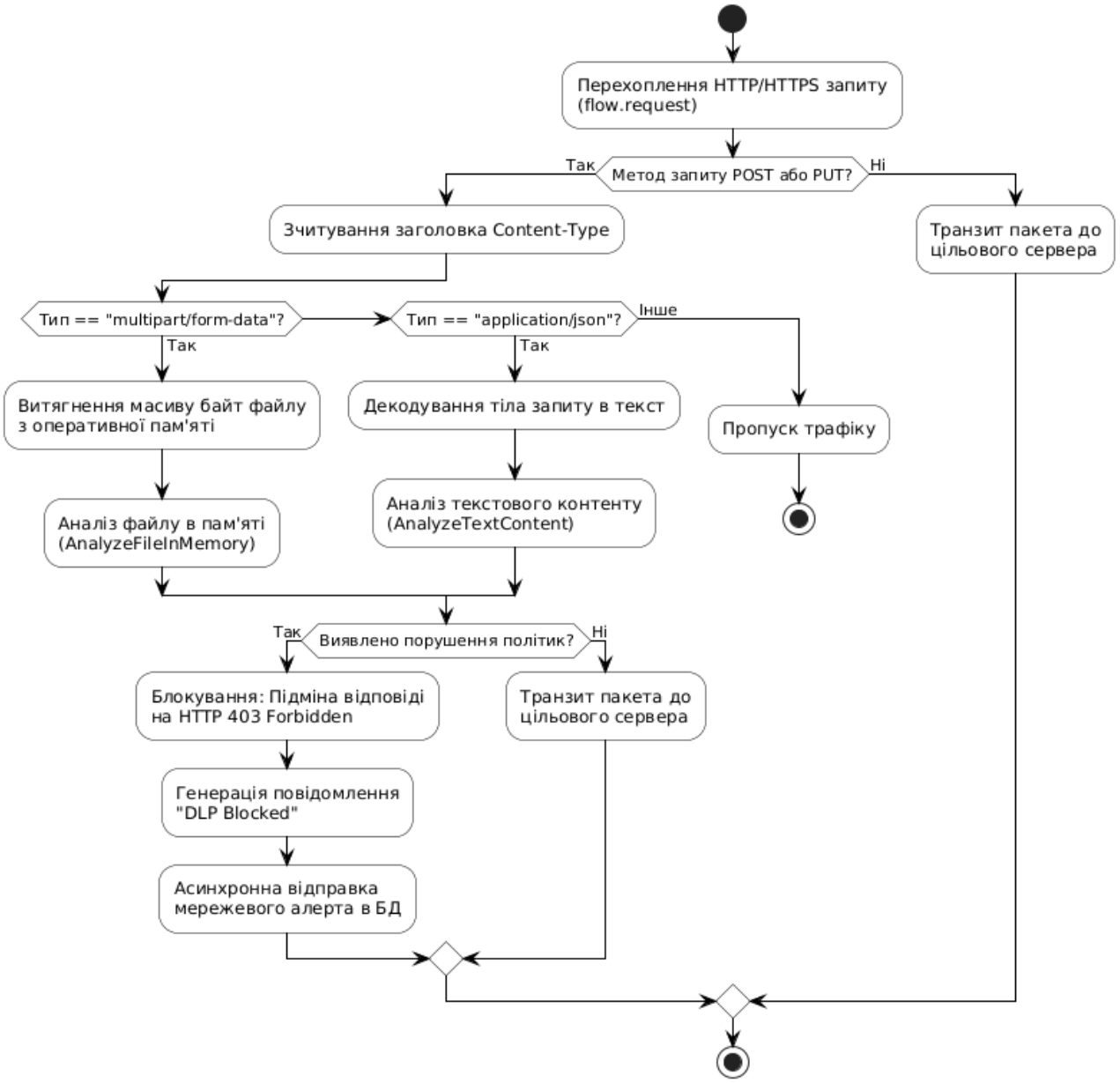


Рисунок 3.4 Алгоритм інспекціх мережевих потоків

3.2 Структура і склад програмного забезпечення

Розроблене програмне забезпечення системи запобігання витокам інформації побудоване за модульним принципом, що дозволяє забезпечити високу гнучкість, масштабованість та простоту супроводу програмного коду. Відповідно до спроєктованої розподіленої архітектури, програмний комплекс фізично та логічно розділений на три незалежні виконувані застосунки, кожен з яких має власний набір залежностей, життєвий цикл та внутрішню структуру.

До складу програмного забезпечення входять такі основні пакети:

1. Пакет програмного забезпечення периферійного агента (скрипт dlp_agent.py).

2. Пакет програмного забезпечення центрального сервера управління (скрипт dlp_server.py та набір вебшаблонів).

3. Пакет програмного забезпечення мережевого шлюзу (скрипт network_interceptor.py).

Усі компоненти розроблено з використанням високорівневої мови програмування Python, що дозволило реалізувати кросплатформенність та використати потужний арсенал вбудованих та сторонніх бібліотек для низькорівневої взаємодії з операційною системою та мережевими протоколами.

Програмний застосунок агента (dlp_agent.py) має яскраво виражену подійно-орієнтовану та багатопотокову структуру. Точка входу в програму, блок __main__ виконує роль ініціалізатора, який зчитує системні змінні, зокрема, ім'я поточного користувача через модуль getpass, формує глобальний словник конфігурації POLICY та запускає пул фонових потоків.

Внутрішня архітектура агента декомпонована на декілька взаємопов'язаних логічних блоків, кожен з яких виконує специфічну роль у загальному конвеєрі обробки даних. Базовим елементом є блок мережевої взаємодії, що включає функції fetch_policies() та send_alert_to_server(). Цей компонент використовує можливості бібліотеки requests для формування HTTP-запитів типів GET та POST, надійно інкапсулюючи комунікаційну логіку та забезпечуючи ізоляцію інших модулів від можливих мережевих збоїв.

Наступним ешелоном виступає блок парсингу та аналізу контенту, центральною функцією якого є extract_text_from_file(). Для глибокої інспекції документів він залучає спеціалізовані сторонні бібліотеки: python-docx для обробки XML-архівів формату Microsoft Word та PyPDF2 для екстракції тексту з бінарних PDF-файлів. При цьому сама логіка пошуку конфіденційних даних

спирається на використання вбудованого модуля re для опрацювання складних регулярних виразів.

Завершальним елементом структурної організації агента є блок криптографічного видалення, у якому функція secure_wipe() безпосередньо взаємодіє з низькорівневими системними інтерфейсами. Для гарантованого знищення інформації вона застосовує виклик модуля операційної системи os.urandom() з метою генерації ентропії, випадкових байтів, а також os.fsync() для примусового апаратного скидання дискових буферів.

Архітектура підсистеми моніторингу реалізована через набір спеціалізованих фонових потоків, кожен з яких використовує відповідні інструментальні засоби для контролю окремого каналу витоку даних. Зокрема, модуль контролю буфера обміну (monitor_clipboard) інтегровано з бібліотекою pyperclip для безперервної роботи з глобальним буфером операційної системи. Паралельно функціонує модуль інспекції знімних носіїв (monitor_removable_media), який залучає можливості бібліотеки psutil для динамічного отримання списку змонтованих дисків шляхом аналізу системних структур disk_partitions.

Своєю чергою, за протидію ручному набору конфіденційної інформації відповідає модуль перехоплення клавіатури, що використовує функціонал бібліотеки keyboard для встановлення глобальних хуків на події системного вводу. Найбільш низькорівневим компонентом є модуль контекстного аналізу вікон, який звертається безпосередньо до Windows API за допомогою бібліотеки ctypes, здійснюючи виклики GetForegroundWindow та GetWindowTextW для отримання дескрипторів активних процесів.

Завдяки такій програмній композиції агент здатний функціонувати як єдиний монолітний застосунок, у якому кожен потоковий модуль ізольовано відповідає за свій специфічний вектор загроз, але водночас усі вони злагоджено синхронізуються, звертаючись до спільного ресурсу конфігурації.

Серверна частина системи розроблена з використанням архітектурного патерну MVC на базі мікрофреймворку Flask. Цей застосунок забезпечує маршрутизацію мережевих запитів та управління персистентним зберіганням даних.

У структурному аспекті серверна частина системи декомпонована на кілька взаємопов'язаних рівнів, що забезпечує чітке розділення відповідальності між її компонентами. Базовим елементом цієї архітектури виступає шар доступу до даних, реалізований за допомогою модуля mysql.connector. Він містить конфігураційний словник DB_CONFIG та функцію get_db_connection(), яка безпосередньо керує пулом з'єднань із системою керування базами даних. У межах цього ж шару функціонує процедура init_db(), призначена для виконання міграцій, автоматичного створення реляційної структури бази даних, зокрема таблиць incidents та policies, а також первинного наповнення системи базовими правилами безпеки за замовчуванням.

Надбудовою над рівнем даних є шар програмного інтерфейсу, який містить набір спеціалізованих маршрутів, призначених для машинної взаємодії між сервером та периферійними агентами. Зокрема, ендпоінт @app.route('/api/policy') здійснює вибірку та форматування даних із бази даних у транзитний формат JSON для їхньої подальшої передачі на кінцеві точки, тоді як маршрут @app.route('/api/incidents') приймає вхідні HTTP POST-запити, десеріалізує вхідне JSON-навантаження та забезпечує надійну фіксацію отриманої телеметрії інцидентів у репозиторії.

Верхнім ешелоном серверної архітектури виступає шар візуалізації та управління, структурований через адміністративні маршрути /admin, /admin/add_policy та /admin/del_policy. Цей рівень використовує можливості вбудованого шаблонізатора Jinja2 для динамічного рендерингу HTML-сторінок.

При цьому сам графічний вебпрезентатор повністю винесено в окремий компонент загальної структури проєкту - файл dashboard.html, у який інтегровано сучасний CSS-фреймворк Bootstrap 5.1 для забезпечення адаптивного дизайну та коректного відображення інтерфейсу на різних типах пристроїв користувачів.

Окремим, але невід'ємним програмним компонентом системи є мережевий шлюз. Архітектурно він розроблений не як самостійний виконуваний файл, а як програмний аддон для фреймворку mitmproxy. Це дозволяє уникнути написання складного низькорівневого коду для парсингу TCP/IP пакетів та зосередитись виключно на логіці безпеки.

Програмна структура мережевого перехоплювача реалізована на базі об'єктно-орієнтованого класу DLPInterceptor, який безпосередньо вбудовується в конвеєр обробки подій проксі-сервера. Життєвий цикл цього компонента починається з методу ініціалізації load, що автоматично викликається базовим фреймворком під час запуску системи. У межах даного методу ініціюється породження ізольованого фонового потоку sync_policies(), який взаємодіє з Flask-сервером та гарантує постійну актуальність словників заборонених слів на мережевому шлюзі. Для безпосередньої інспекції даних використовуються модулі аналізу в оперативній пам'яті, представлені методами analyze_text() та analyze_file_in_memory(). Хоча структурно вони є ідентичними до відповідних аналізаторів локального агента і також залучають бібліотеки docx та PyPDF2, їхня архітектура була суттєво адаптована для роботи з віртуальними байтовими потоками io.BytesIO().

Таке рішення дозволяє здійснювати глибокий розбір файлів на льоту виключно в межах оперативної пам'яті, повністю виключаючи необхідність їхнього проміжного збереження на фізичний жорсткий диск шлюзу. Центральним елементом логіки інспектування виступає головний обробник request(self, flow: http.HTTPFlow), який автоматично перехоплює кожен транзитний HTTP-запит.

Його внутрішня архітектура базується на розгалужених деревах умовних операторів, які забезпечують точну класифікацію типу мережевого контенту. Зокрема, при виявленні заголовка multipart/form-data модуль ініціює парсинг навантаження як процесу завантаження файлу. У випадку ідентифікації порушення політик безпеки, цей метод здійснює структурну модифікацію об'єкта flow.response, примусово повертаючи клієнту HTTP-статус 403 Forbidden, що забезпечує миттєве блокування транзиту несанкціонованого пакета до зовнішньої мережі.

Ефективне функціонування розподіленої системи запобігання витокам інформації неможливе без надійної підсистеми персистентного зберігання даних. Інформаційне забезпечення розробленого програмного комплексу реалізовано на базі реляційної системи управління базами даних MySQL.

Вибір реляційної моделі обґрунтований необхідністю дотримання принципів ACID, що є критичною вимогою для систем аудиту та інформаційної безпеки, де втрата логів інцидентів є неприпустимою.

Логічна та фізична модель бази даних (рис.3.5), яка отримала назву dlp_system, спроєктована за принципом мінімальної достатності та оптимізована для швидкого читання політик безпеки і високочастотного запису інцидентів. Відповідно до спроєктованої схеми, база даних складається з трьох основних нормалізованих таблиць: таблиці конфігурацій безпеки або policies, журналу реєстрації подій або incidents та таблиці облікових записів адміністраторів або admins.

Таблиця policies виступає централізованим сховищем правил, які диктують поведінку всіх периферійних агентів та мережевого шлюзу. На відміну від класичної моделі ключ-значення, структура таблиці передбачає використання окремих стовпців для різних типів правил, що пришвидшує їх вибірку серверним API:

1) id (int(11), PRIMARY KEY) – унікальний сурогатний ідентифікатор набору політик;

2) blocked_extensions (varchar(255)) - перелік розширень файлів, запис або передача яких суворо заборонена, наприклад, виконувані файли чи скрипти;

3) forbidden_words (text) - масив маркерних або секретних слів, наявність яких у тексті ініціює блокування передачі даних;

4) regex_patterns (text) - набір регулярних виразів для виявлення структурованих конфіденційних даних (паролів, номерів кредитних карток, ідентифікаційних кодів).

Таблиця incidents виконує роль центрального журналу аудиту куди асинхронно стікається телеметрія з усіх робочих станцій та шлюзів. Вона спроєктована для збереження детального контексту кожного витоку і містить такі поля:

1) id (int(11), PRIMARY KEY) - унікальний номер інциденту в системі;

2) timestamp (datetime) - точна часова мітка фіксації порушення, яка зберігається у нативному форматі дати і часу для забезпечення можливості подальшого хронологічного сортування та аналітики;

3) username (varchar(255)) - ідентифікатор суб'єкта порушення (ім'я користувача ОС, зафіксоване агентом, або ідентифікатор вузла);

4) violation_type (varchar(255)) - категорія загрози, наприклад, "Оптичний витік даних", "Секретне слово у файлі USB";

5) details (text) - розширений опис інциденту (назва перехопленого файлу, фрагмент надрукованого тексту або контекст вікна);

6) action_taken (varchar(255)) - статус реакції системи на інцидент.

Таблиця admins призначена для забезпечення підсистеми автентифікації та контролю доступу до центральної веб-панелі управління. Вона ізолює керування системою від несанкціонованого втручання:

1) id (int(11), PRIMARY KEY) - ідентифікатор користувача-адміністратора;

2) username (varchar(50)) - логін офіцера інформаційної безпеки;

3) password_hash (varchar(255)) - криптографічний хеш пароля, що забезпечує неможливість компрометації облікових даних навіть у випадку отримання зловмисником прямого доступу до файлів бази даних.

Загальна архітектура інформаційного забезпечення дозволяє системі накопичувати великі обсяги даних без суттєвої деградації швидкодії та легко інтегруватися з BI-системами для побудови аналітичних звітів щодо безпеки.

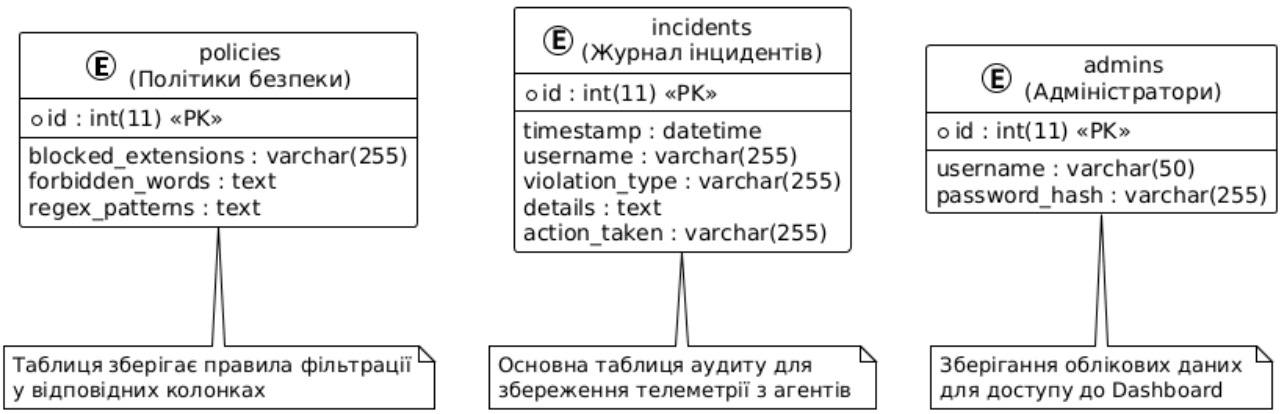


Рисунок 3.5 - Схема бази даних

Загалом, структура програмного забезпечення відповідає принципу єдиної відповідальності. Кожен файл та клас виконує чітко визначену роль: агент відповідає за глибокий моніторинг ОС, інтерсептор - за мережевий трафік, а сервер - за агрегацію даних та надання користувацького інтерфейсу, що створює єдину кіберфізичну екосистему.

3.3 Веб-базований інтерфейс для доступу та управління системою

Ефективність функціонування будь-якої комплексної системи інформаційної безпеки залежить не лише від надійності її математичних алгоритмів виявлення загроз, а й від ергономічності засобів адміністрування. Для забезпечення централізованого управління розподіленими агентами та оперативного моніторингу інцидентів у розробленому програмно-технічному комплексі спроєктовано спеціалізований веб-базований графічний інтерфейс.

Вибір веб-орієнтованої архітектури для побудови людино-машинного інтерфейсу обґрунтований низкою суттєвих переваг порівняно з класичними десктопними застосунками:

- кросплатформенність та відсутність необхідності встановлення додаткового програмного забезпечення на робоче місце офіцера безпеки;
- можливість безпечного віддаленого доступу до панелі управління через захищені канали зв'язку з будь-якого пристрою;
- незалежність життєвого циклу інтерфейсу від ядра системи, що дозволяє оновлювати візуальні компоненти без зупинки процесів моніторингу на кінцевих точках.

Інтерфейс побудовано на базі сучасного каскадного CSS-фреймворку Bootstrap 5.1. Використання цього інструментарію забезпечило реалізацію концепції адаптивного веб-дизайну, завдяки чому панель управління коректно відображається на екранах з різною роздільною здатністю, від мобільних пристроїв до широкоформатних моніторів. Серверна генерація HTML-сторінок здійснюється за допомогою шаблонізатора Jinja2, інтегрованого у фреймворк Flask, що дозволяє динамічно інкапсулювати дані з бази даних MySQL безпосередньо у візуальні компоненти.

Візуальний простір головної сторінки адміністратора логічно розділений на дві функціональні зони, які відповідають за два ключові процеси управління DLP-системою: проактивне налаштування правил (постановка завдань системі) та реактивний моніторинг інцидентів.

Для забезпечення централізованого управління правилами безпеки в системі передбачено спеціалізований вебінтерфейс (рис. 3.6), який надає адміністратору можливість динамічно формувати та редагувати критерії, за якими агенти здійснюють пошук конфіденційної інформації. Цей компонент побудовано на базі HTML-форми, яка під час збереження введених даних ініціює HTTP POST-запит на серверний маршрут /admin/add_policy.

Під час налаштування системи офіцер безпеки має змогу обрати тип нової політики з випадаючого списку, що підтримує три ключові категорії класифікації. Зокрема, категорія “Заборонене слово” дозволяє задавати маркерні терміни, специфічні для конкретного підприємства, наприклад, “гриф секретно” чи “фінансовий звіт”. Своєю чергою, категорія “Заборонене розширення” застосовується для внесення форматів файлів, таких як .exe або .bat, з метою превентивного блокування копіювання чи запуску сторонніх програм зі знімних USB-носіїв. Категорія “Регулярний вираз” передбачає введення складних синтаксичних конструкцій формату RegEx, які необхідні для точного автоматизованого виявлення структурованих конфіденційних даних, наприклад, номерів банківських рахунків або телефонів.

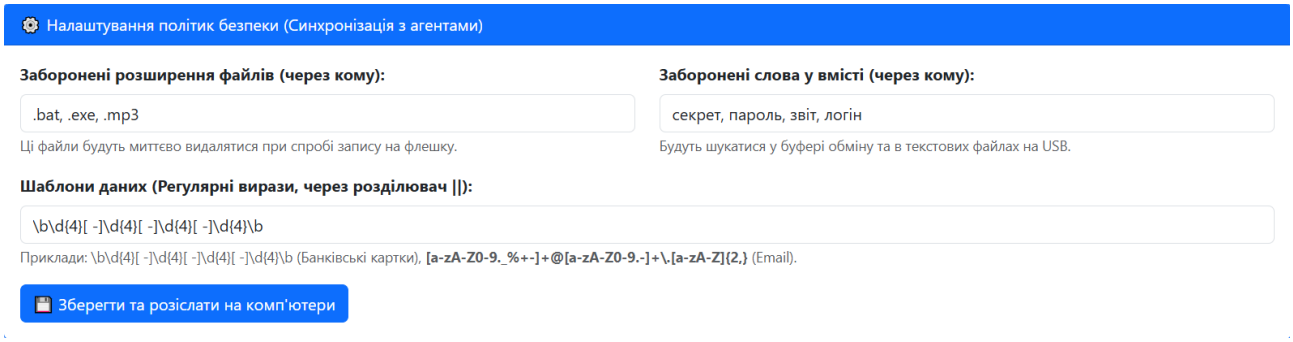


Рисунок 3.6 - Зовнішній вигляд модуля Управлівння політками безпеки

Нижній сегмент робочого простору адміністративного вебінтерфейсу (рис. 3.7) відведено під модуль агрегації та моніторингу інцидентів, головним елементом якого є журнал подій, реалізований у вигляді прокручуваної таблиці з фіксованою висотою. Таке архітектурне рішення дозволяє розмістити великий масив логів на одному екрані, уникаючи необхідності застосування постійної посторінкової навігації. Сама таблиця відображає розгорнуту телеметрію, яку центральний Flask-сервер безперервно агрегує з усіх розподілених компонентів системи.

Для зручності аналізу дані структуровані за ключовими параметрами, серед яких насамперед фіксується хронологічний показник моменту виявлення

витоку та ідентифікатор джерела порушення, яким може виступати ім'я користувача операційної системи або маркер мережевого шлюзу. Також здійснюється детальна категоризація інциденту за типом загрози, наприклад, оптичний витік даних, передача через USB-носій чи контроль клавіатури.

Окрім цього, кожен запис містить специфічний контекст події, що включає заголовок вікна, назву файлу або фрагмент перехопленого тексту, а також відображає зворотний зв'язок від периферійного агента щодо успішності застосування превентивних заходів, зокрема підтвердження алгоритмічного затирання файлу чи примусового завершення процесу.

На серверному рівні логіка динамічного рендерингу цієї таблиці реалізована з використанням циклів шаблонізатора Jinja2. При цьому для оптимізації користувацького досвіду передбачено обробку порожніх станів: у разі відсутності записів у базі даних інтерфейс автоматично генерує повідомлення про те, що інцидентів поки не виявлено.

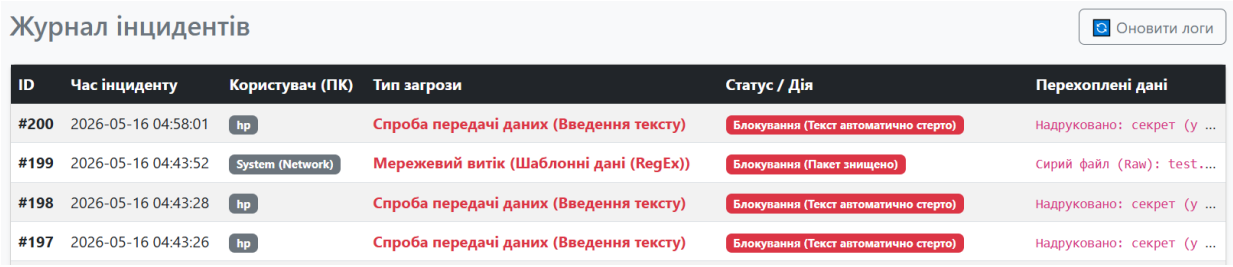


Рисунок 3.7 Зовнішній вигляд модуля агрегації та моніторингу інцидетів

Панель управління розподіленою DLP-системою (рис. 3.8) є критично важливим об'єктом інфраструктури, оскільки отримання зловмисником доступу до неї дозволить повністю нівелювати захист підприємства шляхом несанкціонованого видалення або модифікації політик безпеки. Для запобігання таким загрозам в архітектуру вебзастосунку інтегровано підсистему автентифікації та авторизації, що взаємодіє з таблицею admins бази даних dlp_system.

Програмна реалізація модуля базується на принципі суворої перевірки повноважень користувача перед наданням доступу до будь-якого внутрішнього ресурсу сервера і структурно охоплює три взаємопов'язані рівні безпеки.

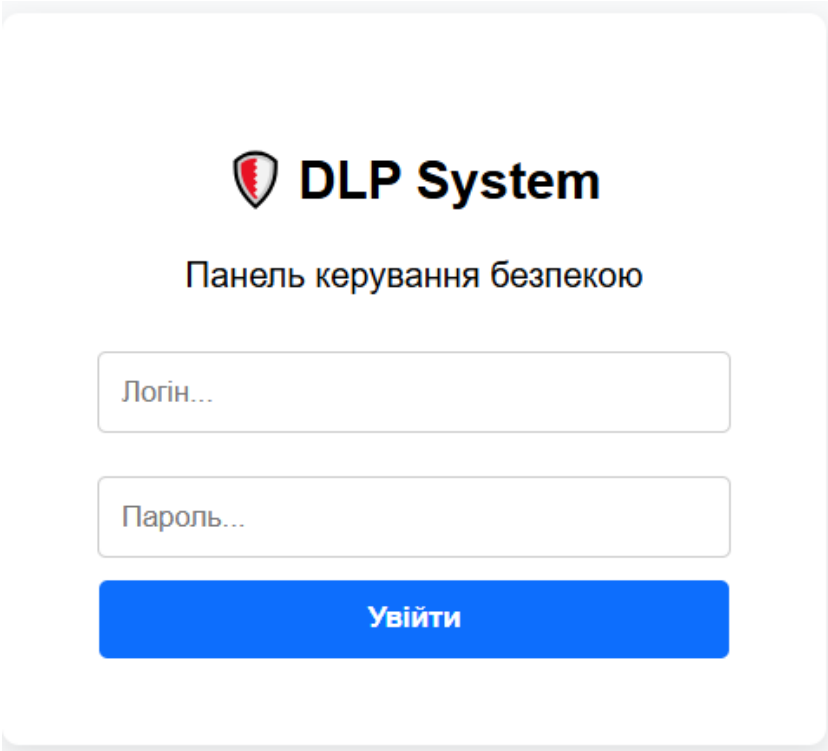


Рисунок 3.8 - Зовнішній вигляд модуля входу

Першим ешелоном виступає криптографічний захист облікових даних, що системно виключає зберігання паролів у відкритому вигляді. Зокрема, під час реєстрації або зміни пароля адміністратора застосовується метод одностороннього криптографічного хешування за алгоритмом PBKDF2 з використанням солі та шифру SHA-256, після чого отриманий рядок фіксується у полі password_hash. Під час спроби входу система зчитує введений користувачем пароль, повторно обчислює хеш із додаванням унікальної солі та порівнює результат із зафіксованим у базі даних, що унеможливлює компрометацію паролів навіть у разі прямого витоку вмісту таблиць СУБД.

Наступним рівнем є керування сесіями: після успішного збігу криптографічних хешів сервер ініціалізує захищену клієнтську сесію за

допомогою механізму flask.session, який зберігає ідентифікатор користувача у зашифрованому cookie-файлі на стороні браузера. Цілісність сесії забезпечується використанням секретного ключа додатка app.secret_key, тому будь-яка спроба підробки або підміни ідентифікатора призводить до її автоматичного анулювання сервером.

Завершальним архітектурним елементом є використання посередників захисту маршрутів. Для обмеження доступу до панелі управління /admin та функціональних API-маршрутів /admin/add_policy і /admin/del_policy реалізовано спеціалізовану логіку, через декоратор @login_required або умову if 'user_id' not in session, яка перехоплює кожен вхідний HTTP-запит до захищеного URL. Якщо у сесії відсутній валідний ідентифікатор адміністратора, виконання цільової функції блокується, а користувач автоматично перенаправляється, через HTTP 302 Redirect на ізольовану сторінку авторизації /login. Лише за умови успішної перевірки прапорця автентифікації запит передається далі за конвеєром Flask для рендерингу робочого простору dashboard.html.

Таким чином, розроблений веб-базований інтерфейс виступає єдиною точкою входу для управління розподіленою системою. Він мінімізує когнітивне навантаження на оператора, надаючи йому лише релевантну інформацію та інтуїтивно зрозумілі інструменти для постановки завдань агентам безпеки.

3.4. Приклади застосування системи

Для верифікації працездатності розробленого програмно-технічного комплексу та оцінки ефективності алгоритмів виявлення загроз було проведено серію імітаційних тестувань, які охоплювали найбільш імовірні вектори витоку конфіденційної інформації в умовах, наближених до реальної експлуатації у корпоративних мережах.

Першим етапом імітаційного тестування стала перевірка ефективності підсистеми запобігання витокам через знімні носії інформації. Для цього було

змодельовано вектор атаки, за якого інсайдер із легітимним доступом до комп'ютера намагається скопіювати на особистий USB-накопичувач фінансовий звіт у форматі .docx, що містить гриф секретно, заздалегідь внесений до політик безпеки.

Реакція системи на цю подію відбувається автоматично: одразу після фізичного підключення флеш-накопичувача до USB-порту робочої станції фоновий потік агента monitor_removable_media за допомогою системних викликів бібліотеки psutil виявляє монтування нового тому. Далі агент розпочинає рекурсивний обхід файлової системи носія і фіксує появу нового файлу фінансовий_звіт.docx.

Для перевірки його вмісту модуль розпаковує XML-структуру документа безпосередньо в оперативній пам'яті з використанням бібліотеки python-docx та проводить семантичний аналіз тексту. Виявивши маркерне слово секретно, алгоритм приймає рішення про негайне блокування передачі даних. Важливо зазначити, що замість простого системного видалення ініціюється виклик функції secure_wipe, яка надійно перезаписує сектори флеш-накопичувача випадковими байтами, унеможливлюючи подальше відновлення документа. Результат успішного блокування витоку за цим вектором проілюстровано на рисунку 3.9.

У результаті цільовий файл гарантовано знищено з носія порушника до того, як він встиг витягнути його з порту. В цей же час адміністратор у веб-панелі управління отримує сповіщення з типом загрози “USB: Секретне слово 'секретно' у файлі” та статусом “Файл затерто та видалено з USB”.

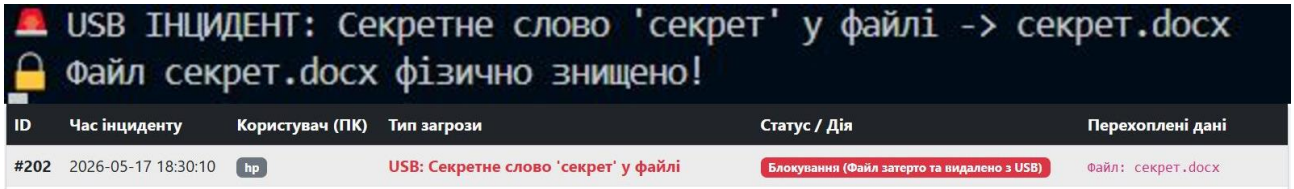


Рисунок 3.9 - Результат блокування запису на USB-носій

Наступним етапом тестування стала перевірка механізмів блокування спроб оптичного перехоплення конфіденційних даних. Для цього було змодельовано ситуацію, за якої користувач легітимно працює з договором, назва якого, наприклад, “секрет.docx” відображається в заголовку активного вікна програми. З метою обходу підсистеми моніторингу буфера обміну, порушник намагається зробити знімок екрана за допомогою стандартної утиліти ОС Windows – “Засіб захоплення фрагментів”.

Протидія цій загрозі відбувається автоматично: фоновий модуль monitor_screenshot_tools із частотою два рази на секунду опитує Windows API і, зафіксувавши, що активним є вікно із забороненим маркерним словом секретно у заголовку, встановлює внутрішній прапорець небезпечного контексту is_confidential_open = True.

Безпосередньо у момент спроби запуску snippingtool.exe агент виявляє появу відповідного процесу в таблиці процесів операційної системи та одразу відправляє йому системний сигнал kill(). Це забезпечує примусове завершення роботи утиліти ще до моменту появи її графічного інтерфейсу на екрані монітора. Приклад успішного блокування витоку інформації за цим вектором наведено на рисунку 3.10.

Як результат користувач фізично не може зробити скриншот, поки на екрані відкритий таємний документ. На сервер відправляється алерт "Оптичний витік даних (Скриншот)".

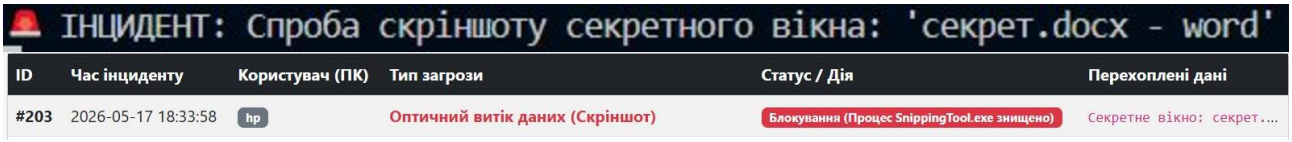


Рисунок 3.10 - Результат роботи блокування скріншоту

Окрему увагу під час тестування було приділено механізмам протидії ручному введенню конфіденційних даних у неконтрольованому середовищі.

Для перевірки цього функціоналу було змодельовано ситуацію, за якої порушник намагається передати номер корпоративної кредитної картки або пароль доступу сторонній особі через вебверсію неконтрольованого месенджера, наприклад, Telegram Web, вручну набираючи текст на клавіатурі.

Протидія цій загрозі реалізується за рахунок роботи фонового потоку monitor_keystrokes, який, використовуючи глобальні Hook-перехоплення подій клавіатури, безперервно формує динамічний кільцевий буфер зі 100 останніх набраних символів. Після кожного фізичного натискання клавіші вміст буфера перевіряється за допомогою набору регулярних виразів, заздалегідь завантажених із центрального сервера.

Як тільки введений рядок збігається із заданим регулярним виразом, система автоматично ідентифікує спробу витоку інформації. З метою активного запобігання несанкціонованій передачі даних агент миттєво генерує серію віртуальних натискань системної клавіші Backspace, кількість яких точно дорівнює довжині забороненого слова, тим самим видаляючи набраний текст з екрана. Приклад успішної роботи цього механізму зворотного стирання тексту наведено на рисунку 3.11

Як результат набраний користувачем конфіденційний текст стирається з екрана месенджера ще до того, як порушник встигне натиснути кнопку відправлення.

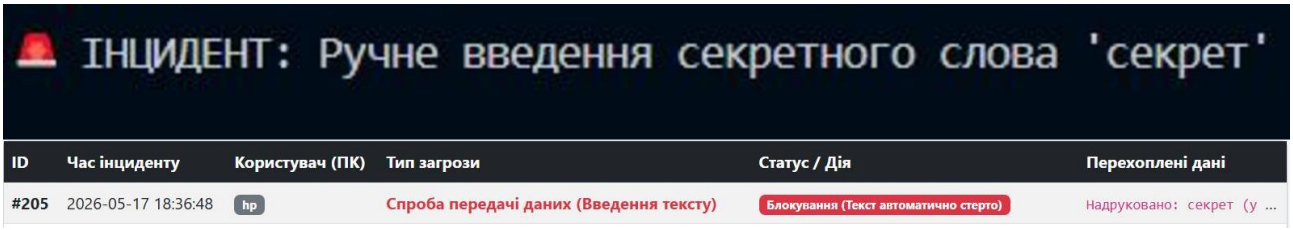


Рисунок 3.11 - Приклад роботи затирання тексту

Завершальним етапом тестування стала перевірка ефективності підсистеми мережевої інспекції, спрямована на блокування передачі даних у зовнішні хмарні середовища.

Для цього було змодельовано ситуацію, за якої користувач із гостьового ноутбука намагається завантажити PDF-документ, що містить фінансову таємницю, на зовнішній хмарний диск через захищене HTTPS-з'єднання.

Протидія цій загрозі здійснюється на рівні маршрутизації: весь вихідний трафік спрямовується через корпоративний шлюз, де функціонує модуль network_interceptor.py на базі фреймворку Mitmproxy. Мережевий перехоплювач динамічно розшифровує TLS-з'єднання, аналізує HTTP-запит і виявляє спробу передачі даних із типом multipart/form-data. Повністю уникаючи проміжного збереження файлу на фізичний жорсткий диск сервера, інтерсептор виділяє бінарні дані у віртуальний потік та екстрагує текстовий вміст PDF-документа в оперативній пам'яті.

Виявивши збіг із забороненим регулярним виразом або маркерним словом, шлюз миттєво перериває транзит пакета до цільового сервера. Натомість він підміняє відповідь, генеруючи для клієнта HTTP-статус 403 Forbidden із системним повідомленням DLP Blocked: Confidential data detected.

Як наслідок, завантаження файлу у хмару успішно блокується на рівні мережі, а у вебпанелі управління автоматично фіксується інцидент із зазначенням джерела події та цільового хоста, на який здійснювалася спроба несанкціонованої передачі. Результат роботи цього механізму перехоплення відображено на рисунку 3.12.

Як результат: Завантаження файлу у хмару заблоковано на рівні мережі. У веб-панелі управління фіксується джерело "Network Gateway" та цільовий хост, на який здійснювалася спроба передачі.

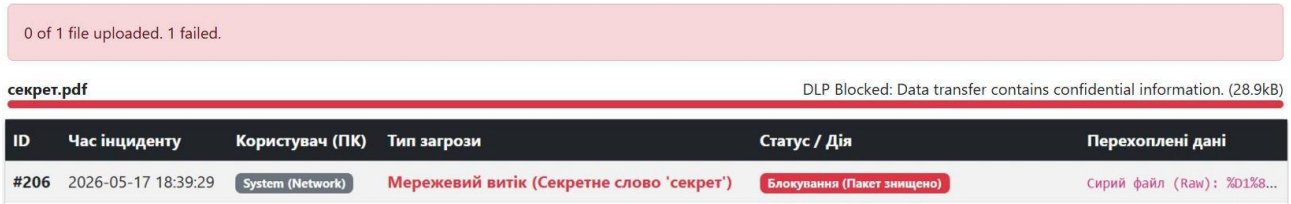


Рисунок 3.12 - Результат роботи мережевого перехоплювача

Для глибокого розуміння динаміки внутрішніх процесів розробленого програмного забезпечення доцільно розглянути логічну модель поведінки системи через аналіз життєвого циклу інциденту. Процес розпочинається з ініціалізації та розгалуження агентів на паралельні потоки, після чого система переходить у суперстан моніторингу. У цьому стані потоки-демони перебувають в оптимізованому режимі пасивного очікування подій операційної системи, споживаючи мінімум ресурсів. Перехід до активної фази перехоплення ініціюється специфічними тригерами: зміною буфера обміну, підключенням знімного носія, натисканням клавіш або надходженням мережевого запиту.

Отримавши тригерний сигнал, конвеєр обробки даних переходить до суперстану аналізу контенту. Цей етап передбачає десеріалізацію об'єктів, вилучення текстової складової з бінарних потоків чи офісних документів та проведення дворівневої перевірки. Спочатку виконується семантичне зіставлення тексту зі словником заборонених маркерів у нижньому регістрі, після чого здійснюється пошук структурованих конфіденційних патернів за допомогою регулярних виразів. На основі результатів цього аналізу відбувається умовне розгалуження: легітимні дії дозволяють системі скинути тимчасові змінні й повернутися до пасивного моніторингу, тоді як виявлення загрози миттєво активує контур реагування.

Суперстан реагування виконує превентивну компенсаційну функцію, ізолюючи вектор атаки та блокуючи подальше розповсюдження інформації. Залежно від джерела загрози, система застосовує активні деструктивні заходи: ініціює гарантоване затирання файлів на носіях, примусово завершує процеси утиліт зняття екрана або генерує зворотні сигнали клавіатури для стирання введеного тексту. Після успішного придушення витоку управління передається стану реєстрації інциденту. Локальний потік агрегує метадані порушення, формує структурований пакет даних та асинхронно відправляє його на центральний сервер управління.

Після підтвердження запису телеметрії у реляційну базу даних транзакція закривається, і розподілені компоненти системи повертаються до початкового стану очікування нових загроз.

3.4 Висновок до третього розділу

У межах третього розділу здійснено практичну програмну реалізацію спроєктованої комп’ютерної системи захисту конфіденційної інформації від витоків , розроблено її алгоритмічне забезпечення, зокрема алгоритми гарантованого затирання файлів, перехоплення клавіатури та інспекції мережевих потоків в оперативній пам'яті , здійснено опис модульної структури програмного забезпечення периферійного агента, центрального сервера управління та мережевого шлюзу.

Крім цього, спроєктовано веббазований інтерфейс для управління політиками безпеки та моніторингу інцидентів , описано фізичну модель реляційної бази даних та проведено імітаційне тестування ефективності розробленого програмно-технічного комплексу за ключовими векторами загроз.

ВИСНОВКИ

У кваліфікаційній роботі за результатами виконаних теоретичних та практичних досліджень було успішно вирішено актуальне науково-технічне завдання щодо проєктування, алгоритмізації та програмної реалізації розподіленої універсальної системи запобігання витокам інформації. Розроблений програмно-техничний комплекс забезпечує надійний захист конфіденційних даних організації в умовах сучасного ландшафту кіберзагроз, мінімізуючи вплив людського фактора та оптимізуючи процеси розслідування інцидентів безпеки.

У першому розділі роботи проведено всебічний аналіз предметної області захисту інформації від витоків, який виявив стрімку ерозію класичного поняття корпоративного мережевого контуру внаслідок масового впровадження хмарних обчислень та гібридних моделей роботи. Досліджено новітні загрози, пов'язані з використанням систем генеративного штучного інтелекту та великих мовних моделей як нових прихованих каналів витоку даних. Шляхом порівняльного аналізу існуючих закордонних рішень таких як: Symantec, Microsoft Purview, Forcepoint, Netskope, Proofpoint було визначено їхні ключові недоліки, такі як висока вартість, складність адміністрування, проблема хибнопозитивних спрацювань та відсутність належної підтримки лінгвістичних особливостей української мови й вимог національного законодавства, зокрема нормативних документів системи ТЗІ. На основі виявлених проблем було сформувано тактико-технічні вимоги та поставлено задачу розробки розподіленої інтелектуальної системи на основі периферійного контентного аналізу.

У другому розділі здійснено проєктування гнучкої трирівневої архітектури розподіленої універсальної системи безпеки, яка складається з підсистеми централізованого управління, підсистеми моніторингу кінцевих точок та підсистеми мережевого інспектування. Обґрунтовано доцільність впровадження парадигми периферійних обчислень, що дозволило перенести основне

обчислювальне навантаження з аналізу контенту безпосередньо на робочі станції користувачів, суттєво знижуючи затримки та навантаження на пропускну здатність мережі. Математично та інженерно доведено необхідність розпаралелювання процесів на кінцевих точках згідно із законом Амдала. Виділення модулів контролю буфера обміну, глибокої інспекції USB-носіїв, моніторингу черг оптичних дисків, перехоплення клавіатури, блокування скриншотів та мережевої синхронізації в ізольовані фонові потоки-демони дозволило повністю нівелювати проблему блокування системи операціями введення-виведення. Крім того, досліджено механізми самоорганізації архітектури, які забезпечують переміщення логічного центру прийняття рішень та збереження захисних функцій компонентів у автономному режимі у випадку тимчасової втрати мережевого зв'язку з сервером управління відповідно до положень CAP-теореми.

У третьому розділі виконано практичну програмно-апаратну реалізацію розробленого комплексу на базі мови програмування Python та СУБД MySQL. Спроектовано та описано високоефективні алгоритми, серед яких: алгоритм гарантованого низькорівневого затирання файлів за допомогою випадкових чисел та системного виклику синхронізації диска, подійно-орієнтований алгоритм зворотного стирання тексту при спробах ручного введення таємниці, а також алгоритм швидкої глибокої інспекції мережевих потоків в оперативній пам'яті без збереження на жорсткий диск шлюзу. Розроблено фізичну модель бази даних dlp_system у складі трьох нормалізованих таблиць, призначених для збереження політик, журналювання інцидентів безпеки та автентифікації адміністраторів. Створено людино-машинний веб-інтерфейс управління на базі фреймворку Flask та CSS-компонентів Bootstrap, який захищено надійним криптографічним модулем авторизації з використанням хешування паролів та механізмів сесій. Проведена серія імітаційних випробувань для різних сценаріїв загроз повністю підтвердила високу точність, швидкодію та відмову від помилкових спрацювань розробленої системи захисту від витоків інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Cost of a Data Breach Report 2024 / IBM Security. Armonk : IBM Corporation, 2024. 46 p. URL: https://wp.table.media/wp-content/uploads/2024/07/30132828/Cost-of-a-Data-Breach-Report-2024.pdf (дата звернення: 09.05.2026).
2. 2024 Data Breach Investigations Report (DBIR) / Verizon. Basking Ridge : Verizon Business, 2024. 100 p. URL: https://www.verizon.com/business/resources/reports/2024-dbir-data-breach-investigations-report.pdf (дата звернення: 09.05.2026).
3. ENISA Threat Landscape 2023 / ENISA. [S. l.] : European Union Agency for Cybersecurity, 2023. 161 p. URL: https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Threat%20Landscape%202023.pdf (дата звернення: 09.05.2026).
4. Microsoft Digital Defense Report 2023 / Microsoft. Redmond : Microsoft, 2023. 131 p. URL: https://www.microsoft.com/en-us/security/security-insider/threat-landscape/microsoft-digital-defense-report-2023 (дата звернення: 09.05.2026).
5. Abid N. Advancements and best practices in data loss prevention: A comprehensive review. *Global Journal of Universal Studies*. 2024. Vol. 1, No. 1. P. 190–225.
6. Yadav I., Gupta H. Designing Data Loss Prevention System for The Enhancement of Data Integrity in Cyberspace. *2023 5th International Conference on Advances in Computing, Communication Control and Networking* (ICAC3N). [S. l.] : IEEE, 2023. P. 1361–1365.
7. 2024 Cybersecurity Readiness Index / Cisco. San Jose : Cisco Systems, 2024. 30 p. URL: https:/i/newsroom.cisco.com/c/dam/r/newsroom/en/us/interactive/cybersecurity-readiness-index/documents/Cisco_Cybersecurity_Readiness_Index_FINAL.pdf (дата звернення: 09.05.2026).

8. National Cybersecurity Strategy / The White House. Washington, D.C. : The White House, 2023. 39 p. URL: https://bidenwhitehouse.archives.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf (дата звернення: 09.05.2026).

9. Data Breach Reporting Requirements / FCC. Washington, D.C. : Federal Communications Commission, 2024. 37 p. URL: https://www.govinfo.gov/content/pkg/FR-2024-02-12/pdf/2024-01667.pdf (дата звернення: 09.05.2026).

10. Cross-Sector Cybersecurity Performance Goals / CISA. Washington, D.C. : Cybersecurity and Infrastructure Security Agency, 2023. 27 p. URL: https://www.cisa.gov/sites/default/files/2023-03/CISA_CPG_REPORT_v1.0.1_FINAL.pdf (дата звернення: 09.05.2026).

11. Top Threats to Cloud Computing Deep Dive 2025 / Cloud Security Alliance. [S. l.] : Cloud Security Alliance, 2025. 49 p. URL: https://dayblinkconsulting.com/wp-content/uploads/2025/04/Top-Threats-to-Cloud-Computing-Deep-Dive-20250421.pdf (дата звернення: 09.05.2026).

12. Dhillon G., Smith K., Dissanayaka I. Information systems security research agenda: Exploring the gap between research and practice. *The Journal of Strategic Information Systems*. 2021. Vol. 30, No. 4. P. 101693..

13. Computational Intelligence for Cybersecurity Management and Applications / ed. Y. Maleh, M. Alazab, S. Mounir. Boca Raton : CRC Press, 2023. 248 p.

14. Cyber Security: Critical Infrastructure Protection / ed. M. Lehto, P. Neittaanmäki. Cham : Springer International Publishing, 2022. 484 p.

15. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection - Information security management systems - Requirements. Geneva : ISO, 2022. 19 p.

16. Center for Internet Security (CIS) Controls Version 8 / CIS. East Greenbush : CIS, 2021. 93 p. URL: https://ftp.kr-labs.com.ua/books/CIS_Controls_v8_Guide.pdf (дата звернення: 09.05.2026).

17. COBIT 2019 Framework: Introduction and Methodology / ISACA. Schaumburg : ISACA, 2019. 64 p. URL: https://rms.koenig-solutions.com/Sync_data/Trainer/QMS/1827-2022414376-Cobit2019.pdf (дата звернення: 09.05.2026).

18. ISO/IEC 27002:2022. Information security, cybersecurity and privacy protection - Information security controls. Geneva : ISO, 2022. 152 p.

19. NIST SP 800-53 Revision 5. Security and Privacy Controls for Information Systems and Organizations. Gaithersburg : NIST, 2020. 492 p.

20. NIST SP 800-207. Zero Trust Architecture. Gaithersburg : National Institute of Standards and Technology, 2020. 59 p.

21. PCI DSS v4.0. Payment Card Industry Data Security Standard. PCI Security Standards Council, 2022. 360 p.

22. Guidelines 01/2021 on Examples regarding Data Breach Notification / European Data Protection Board (EDPB). Brussels : EDPB, 2021. 32 p. URL: https://www.edpb.europa.eu/system/files/2022-01/edpb_guidelines_012021_pdbnotification_adopted_en.pdf (дата звернення: 09.05.2026).

23. Regulation (EU) 2024/1689 of the European Parliament and of the Council (Artificial Intelligence Act) . Official Journal of the European Union . 2024. 144 p. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L_202401689 (дата звернення: 09.05.2026).

24. Phiayura P., Teerakanok S. A Comprehensive Framework for Migrating to Zero Trust Architecture . IEEE Access . 2023. Vol. 11. P. 19488–19495.

25. Prasad P. S. S., Nayak S. K., Krishna M. V. Enhancing Insider Threat Detection with Machine Learning Techniques. *2024 International Conference on*

Integrated Intelligence and Communication Systems (ICIICS). [S. l.] : IEEE, 2024. P. 1–5.

26. Alzaabi F. R., Mehmood A. A review of recent advances, challenges, and opportunities in malicious insider threat detection using machine learning methods. *IEEE Access*. 2024. Vol. 12. P. 30907–30927

27. Kornienko M. B. et al. Detecting and Identifying Insider Threats Based on Advanced Clustering Methods . IEEE Access . 2024. Vol. 12. P. 30242–30253.

28. Roy K. C., Chen G. GraphCH: A Deep Framework for Assessing Cyber-Human Aspects in Insider Threat Detection . IEEE Transactions on Dependable and Secure Computing . 2024. Vol. 21, No. 5. P. 4495–4509.

29. Kumar R. Thee machine learning analysis of data granularity for insider threat detection. 2023 *4th IEEE Global Conference for Advancement in Technology (GCAT).* [S. l.] : IEEE, 2023. P. 1–7.

30. Tian T. et al. Insider threat detection for specific threat scenarios . Cybersecurity . 2025. Vol. 8, No. 1. P. 17.

31. Bhargava M. et al. Hunting for Insider Threats Using LSTM-Based Anomaly Detection . IEEE Transactions on Dependable and Secure Computing . 2023. Vol. 20, No. 1. P. 451–462.

32. Perisic J. et al. Sentiment classification for insider threat identification using metaheuristic optimized machine learning classifiers . Scientific Reports . 2024. Vol. 14, No. 1. P. 25731.

33. Bin Sarhan B., Altwaijry N. Insider threat detection using machine learning approach. *Applied Sciences*. 2022. Vol. 13, No. 1. P. 259.

34. Al-Mhiqani M. et al. A Review of Insider Threat Detection: Classification, Machine Learning Techniques, Datasets, Open Challenges, and Recommendations . Applied Sciences . 2020. Vol. 10, No. 15. P. 5208.

35. Yousef R., Jazzar M., Eleyan A., Bejaoui T. A machine learning framework & development for insider cyber-crime threats detection. *2023*

International Conference on Smart Applications, Communications and Networking (SmartNets). [S. l.] : IEEE, 2023. P. 1–6.

36. Lallie H. S. et al. Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic . Computers & Security . 2021. Vol. 105. P. 102248.

37. Politou E., Alepis E., Virvou M., Patsakis C. Privacy and data protection challenges in the distributed era. Vol. 26. Cham : Springer, 2022. 185 p.

38. Rauf U., Wei Z., Mohsen F. Employee Watcher: A Machine Learning-based Hybrid Insider Threat Detection Framework . 2023 7th Cyber Security in Networking Conference (CSNet) . *IEEE*, 2023. P. 39–45.

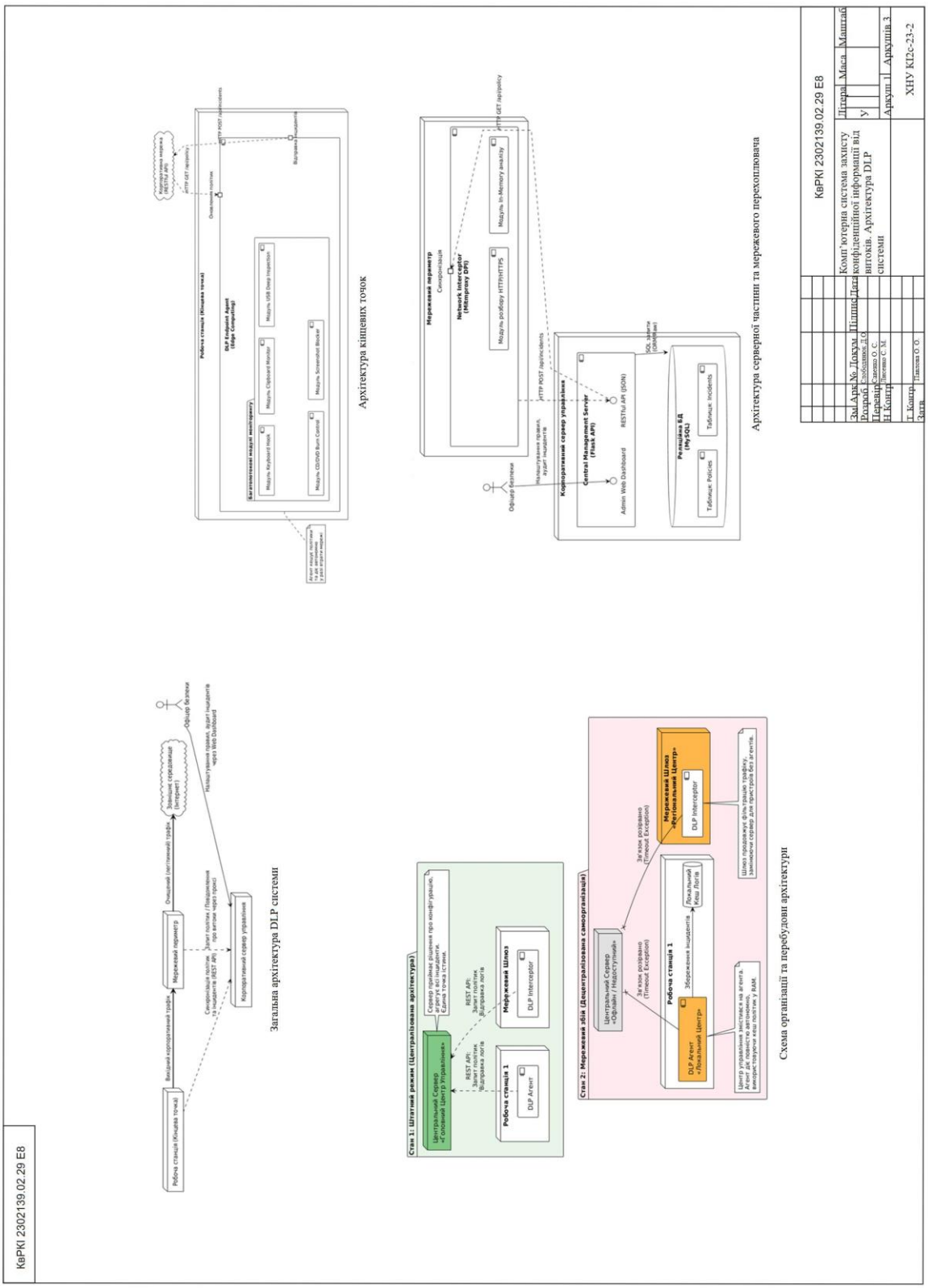
39. Yuan S., Wu X. Deep learning for insider threat detection: Review, challenges and opportunities . *Computers & Security* . 2021. Vol. 104. P. 102221.

40. Nimbalkar P., Kshirsagar D. Feature selection for intrusion detection system in Internet-of-Things (IoT) . *ICT Express* . 2021. Vol. 7, No. 2. P. 177–181.

ДОДАТОК А

(обов'язковий)

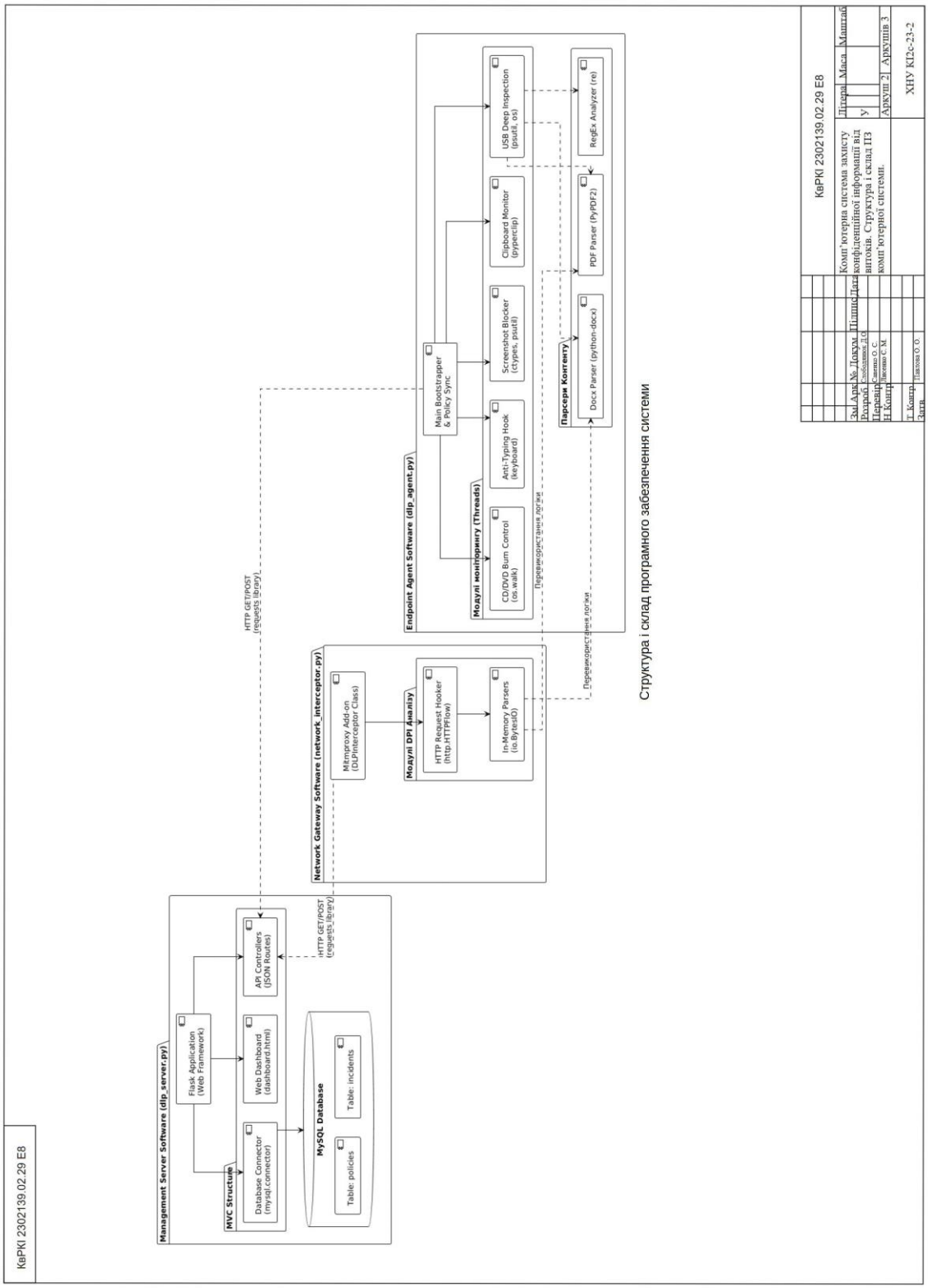
Копія креслення «Архітектура DLP системи»



ДОДАТОК Б

(обов'язковий)

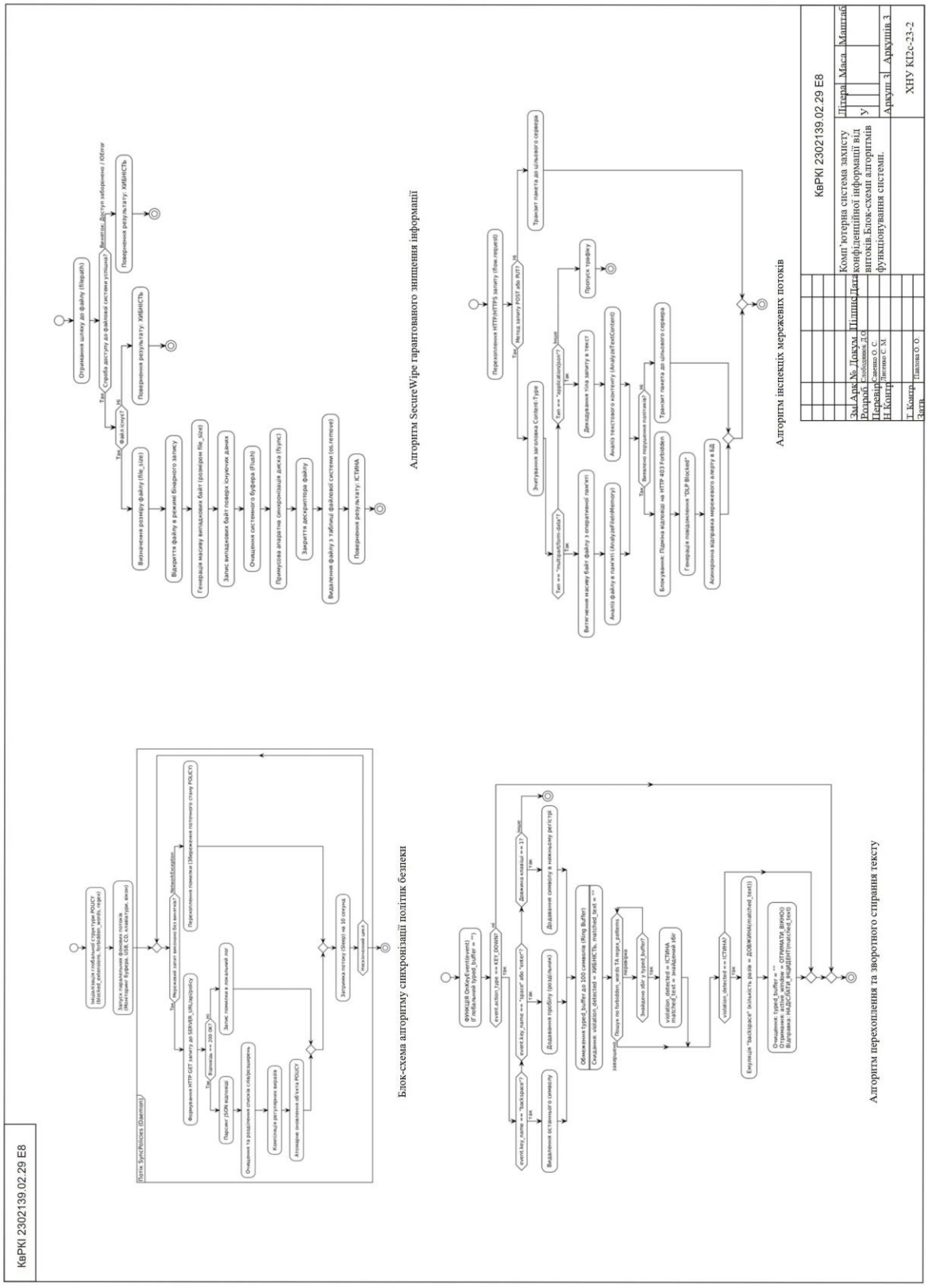
Копія креслення «Структура і склад ПЗ комп'ютерної системи»



ДОДАТОК В

(обов’язковий)

Копія креслення «Блок-схеми алгоритмів функціонування системи»



ДОДАТОК Г

Лістинг програмного коду

dlp_agent.py

```
import pyperclip
import time
import re
import requests
import getpass
import psutil
import os
import threading
import ctypes
import keyboard
from datetime import datetime
from docx import Document
from PyPDF2 import PdfReader

SERVER_BASE_URL = "http://127.0.0.1:5000"
CURRENT_USER = getpass.getuser()

API_SECRET_KEY = "SuperSecretEnterpriseKey2026"

POLICY = {
    "blocked_extensions": [],
    "forbidden_words": [],
    "regex_patterns": []
}

def fetch_policies():
    """Фоновий потік: завантажує нові правила з сервера кожні 10 секунд"""
    global POLICY
    print("[*] Модуль: Синхронізація політик з сервером - АКТИВОВАНО")

    while True:
        try:
            response = requests.get(f"{SERVER_BASE_URL}/api/policy",headers={'X-API-Key': API_SECRET_KEY},
timeout=3)
            if response.status_code == 200:
                data = response.json()

                if 'blocked_extensions' in data and data['blocked_extensions']:
                    POLICY['blocked_extensions'] = [x.strip().lower() for x in data['blocked_extensions'].split(',') if x.strip()]
                else: POLICY['blocked_extensions'] = []
```

```
                    if 'forbidden_words' in data and data['forbidden_words']:
                        POLICY['forbidden_words'] = [x.strip().lower() for x in data['forbidden_words'].split(',') if x.strip()]
                    else: POLICY['forbidden_words'] = []

                    if 'regex_patterns' in data and data['regex_patterns']:
                        raw_patterns = data['regex_patterns'].split('||')
                        compiled_patterns = []
                        for p in raw_patterns:
                            p = p.strip()
                            if p:
                                try: compiled_patterns.append(re.compile(p))
                                except: pass
                        POLICY['regex_patterns'] = compiled_patterns
                    else: POLICY['regex_patterns'] = []
                except:
                    pass
                time.sleep(10)

            def send_alert_to_server(violation_type, intercepted_text, action):
                """Відправляє лог інциденту на центральний сервер"""
                payload = {
                    "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
                    "user": CURRENT_USER,
                    "violation_type": violation_type,
                    "details": intercepted_text,
                    "action_taken": action
                }
                try: requests.post(f"{SERVER_BASE_URL}/api/incidents", json=payload, headers={'X-API-Key': API_SECRET_KEY},
timeout=3)
                except: pass

            def extract_text_from_file(filepath, ext):
                """Витягує чистий текст із різних форматів файлів (TXT, DOCX, PDF)"""
                extracted_text = ""
                try:
                    if ext in ['.txt', '.csv', '.log', '.md']:
                        with open(filepath, 'r', encoding='utf-8', errors='ignore') as f:
                            extracted_text = f.read()
                    elif ext == '.docx':
                        doc = Document(filepath)
                        extracted_text = "\n".join([paragraph.text for paragraph in doc.paragraphs])
                    elif ext == '.pdf':
                        reader = PdfReader(filepath)
                        for page in reader.pages:
                            text = page.extract_text()
                            if text: extracted_text += text
```

```
    except Exception as e:
        print(f"[!] Не вдалося прочитати вміст файлу {filepath}: {e}")
    return extracted_text.lower()

def secure_wipe(filepath):
    """Фізично перезаписує файл випадковими даними перед видаленням (DOD 5220.22-M)"""
    try:
        file_size = os.path.getsize(filepath)
        with open(filepath, "rb+") as f:
            f.write(os.urandom(file_size))
            f.flush()
            os.fsync(f.fileno())
        os.remove(filepath)
        return True
    except Exception:
        return False

def get_active_window_title():
    """Отримує заголовок вікна, яке зараз відкрито у користувача на передньому плані"""
    try:
        hwnd = ctypes.windll.user32.GetForegroundWindow()
        length = ctypes.windll.user32.GetWindowTextLengthW(hwnd)
        buf = ctypes.create_unicode_buffer(length + 1)
        ctypes.windll.user32.GetWindowTextW(hwnd, buf, length + 1)
        return buf.value
    except:
        return "Невідоме вікно"

def monitor_clipboard():
    """Перевіряє скопійований текст"""
    print("[*] Модуль: Контроль буфера обміну - АКТИВОВАНО")
    last_clipboard = ""
    while True:
        try:
            current = pyperclip.paste()
            if current != last_clipboard and current != "":
                current_lower = current.lower()
                violation = None

                for word in POLICY.get('forbidden_words', []):
                    if word in current_lower:
                        violation = f"Заборонене слово '{word}' у буфері"
                        break

                if not violation:
                    for pattern in POLICY.get('regex_patterns', []):
```

```
                if pattern.search(current):
                    violation = f"Виявлено шаблонні дані (RegEx) у буфері"
                    break

            if violation:
                print(f"\n🚨 ІНЦИДЕНТ: {violation}")
                pyperclip.copy("")
                last_clipboard = ""
                send_alert_to_server(violation, current, "Блокування (Буфер очищено)")
            else:
                last_clipboard = current
        except: pass
        time.sleep(1)

def monitor_removable_media():
    """Моніторить флешки та глибоко аналізує файли"""
    print("[*] Модуль: Розумний контроль USB (Deep Inspection) - АКТИВОВАНО")
    known_files = {}

    while True:
        try:
            current_drives = [p.mountpoint for p in psutil.disk_partitions() if 'removable' in p.opts or p.device in ['A:\\', 'B:\\']]
            for drive in current_drives:
                if drive not in known_files:
                    known_files[drive] = set()
                    if os.path.exists(drive):
                        for root, _, files in os.walk(drive):
                            for f in files: known_files[drive].add(os.path.join(root, f))

                if os.path.exists(drive):
                    for root, _, files in os.walk(drive):
                        for file in files:
                            filepath = os.path.join(root, file)
                            if filepath not in known_files[drive]:
                                filename = os.path.basename(filepath)
                                ext = os.path.splitext(filename)[1].lower()
                                violation = None

                                if ext in POLICY.get('blocked_extensions', []):
                                    violation = f"Заборонене розширення ({ext})"
                                elif ext in ['.txt', '.csv', '.log', '.md', '.docx', '.pdf']:
                                    time.sleep(1)
                                    content_lower = extract_text_from_file(filepath, ext)

                                    if content_lower:
                                        for word in POLICY.get('forbidden_words', []):
```

```
                        if word in content_lower:
                            violation = f"Секретне слово '{word}' у файлі"
                            break
                    if not violation:
                        for pattern in POLICY.get('regex_patterns', []):
                            if pattern.search(content_lower):
                                violation = f"Шаблонні дані (RegEx) у файлі"
                                break

                if violation:
                    print(f"🚨 USB ІНЦИДЕНТ: {violation} -> {filename}")
                    if secure_wipe(filepath):
                        action_msg = "Блокування (Файл затерто та видалено з USB)"
                        print(f"🔒 Файл {filename} фізично знищено!")
                    else:
                        action_msg = "Помилка (Не вдалося затерти файл)"
                    send_alert_to_server(f"USB: {violation}", f"Файл: {filename}", action_msg)

                known_files[drive].add(filepath)

        for drive in list(known_files.keys()):
            if drive not in current_drives:
                del known_files[drive]
    except: pass
    time.sleep(2)

def monitor_cd_burning():
    """Моніторить приховану папку підготовки файлів для запису на диски"""
    print("[*] Модуль: Контроль оптичних дисків (CD/DVD Burn) - АКТИВОВАНО")
    local_app_data = os.environ.get('LOCALAPPDATA', '')
    if not local_app_data: return

    burn_folder = os.path.join(local_app_data, 'Microsoft', 'Windows', 'Burn')
    known_burn_files = set()

    while True:
        try:
            if os.path.exists(burn_folder):
                for root, _, files in os.walk(burn_folder):
                    for file in files:
                        filepath = os.path.join(root, file)
                        if filepath not in known_burn_files:
                            filename = os.path.basename(filepath)
                            ext = os.path.splitext(filename)[1].lower()
                            violation = None
```

```
                if ext in POLICY.get('blocked_extensions', []):
                    violation = f"Заборонене розширення ({ext})"
                elif ext in ['.txt', '.csv', '.log', '.md', '.docx', '.pdf']:
                    time.sleep(1)
                    content_lower = extract_text_from_file(filepath, ext)
                    if content_lower:
                        for word in POLICY.get('forbidden_words', []):
                            if word in content_lower:
                                violation = f"Секретне слово '{word}' у файлі"
                                break
                        if not violation:
                            for pattern in POLICY.get('regex_patterns', []):
                                if pattern.search(content_lower):
                                    violation = f"Шаблонні дані (RegEx) у файлі"
                                    break

                if violation:
                    print(f"🚨 CD/DVD ІНЦИДЕНТ: {violation} -> {filename}")
                    if secure_wipe(filepath):
                        action_msg = "Блокування (Файл видалено з черги запису CD/DVD)"
                        print(f"💿 Спроба запису на диск зупинена! {filename} знищено.")
                    else:
                        action_msg = "Помилка (Не вдалося затерти файл)"
                    send_alert_to_server(f"CD/DVD: {violation}", f"Файл: {filename}", action_msg)

                known_burn_files.add(filepath)

            current_files_on_disk = set()
            if os.path.exists(burn_folder):
                for root, _, files in os.walk(burn_folder):
                    for f in files: current_files_on_disk.add(os.path.join(root, f))

            known_burn_files.intersection_update(current_files_on_disk)
        except: pass
        time.sleep(2)

def monitor_screenshot_tools():
    """Фоновий потік: Блокує інструменти створення скріншотів, якщо відкритий таємний документ"""
    print("[*] Модуль: Розумне блокування скріншотів (Context-Aware) - АКТИВОВАНО")

    forbidden_processes = [
        "snippingtool.exe",
        "screenclippinghost.exe",
        "lightshot.exe",
        "sharex.exe"
    ]
```

```
    while True:
        try:
            active_title = get_active_window_title().lower()
            is_confidential_open = False

            for word in POLICY.get('forbidden_words', []):
                if word and word in active_title:
                    is_confidential_open = True
                    break

            if is_confidential_open:
                for proc in psutil.process_iter(['name']):
                    if proc.info['name'] and proc.info['name'].lower() in forbidden_processes:
                        proc.kill()
                        print(f" ІНЦИДЕНТ: Спроба скріншоту секретного вікна: '{active_title}'")
                        send_alert_to_server(
                            "Оптичний витік даних (Скріншот)",
                            f"Секретне вікно: {active_title}",
                            f"Блокування (Процес {proc.info['name']} знищено)"
                        )
        except Exception: pass
        time.sleep(0.5)

def monitor_keystrokes():
    """Фоновий потік: Перехоплює натискання клавіш та стирає секретні слова"""
    print("[*] Модуль: Контроль клавіатури (Anti-Typing) - АКТИВОВАНО")

    typed_buffer = ""

    def on_key_event(e):
        nonlocal typed_buffer

        if e.event_type == keyboard.KEY_DOWN:
            if e.name == 'backspace':
                typed_buffer = typed_buffer[:-1]
            elif e.name in ['space', 'enter']:
                typed_buffer += " "
            elif len(e.name) == 1:
                typed_buffer += e.name.lower()

            if len(typed_buffer) > 100:
                typed_buffer = typed_buffer[-100:]

            violation = None
            matched_text = ""
```

```
            for word in POLICY.get('forbidden_words', []):
                if word and word in typed_buffer:
                    violation = f"Ручне введення секретного слова '{word}'"
                    matched_text = word
                    break

            if not violation:
                for pattern in POLICY.get('regex_patterns', []):
                    match = pattern.search(typed_buffer)
                    if match:
                        violation = "Ручне введення шаблонних даних (RegEx)"
                        matched_text = match.group()
                        break

            if violation:
                print(f"\n ІНЦИДЕНТ: {violation}")


                for _ in range(len(matched_text)):
                    keyboard.send('backspace')

                typed_buffer = ""

                active_window = get_active_window_title()
                send_alert_to_server(
                    "Спроба передачі даних (Введення тексту)",
                    f"Надруковано: {matched_text} (у вікні: {active_window})",
                    "Блокування (Текст автоматично стерто)"
                )

        keyboard.hook(on_key_event)
        while True:
            time.sleep(1)

    if __name__ == "__main__":
        print(f"==========================================")
        print(f"🛡️ Enterprise DLP Agent | Користувач: {CURRENT_USER}")
        print(f"==========================================")

        threading.Thread(target=fetch_policies, daemon=True).start()
        threading.Thread(target=monitor_clipboard, daemon=True).start()
        threading.Thread(target=monitor_removable_media, daemon=True).start()
        threading.Thread(target=monitor_cd_burning, daemon=True).start()
        threading.Thread(target=monitor_screenshot_tools, daemon=True).start()
        threading.Thread(target=monitor_keystrokes, daemon=True).start()
```

```
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print("\n[!] Зупинка DLP Агента...")
```

dlp_server.py

```
import mysql.connector
from flask import Flask, request, jsonify, render_template, redirect, abort, session, url_for
from werkzeug.security import generate_password_hash, check_password_hash
from waitress import serve

app = Flask(__name__)

DB_CONFIG = {
    'host': '127.0.0.1',
    'user': 'root',
    'password': '',
    'database': 'dlp_system'
}


app.config['SECRET_KEY'] = 'SuperSecretKeyForSessions_123987'
API_SECRET_KEY = "SuperSecretEnterpriseKey2026"

def get_db_connection():
    return mysql.connector.connect(**DB_CONFIG)


def init_default_admin():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT COUNT(*) FROM admins")
        if cursor.fetchone()[0] == 0:
            hashed_password = generate_password_hash("admin")
            cursor.execute(
                "INSERT INTO admins (username, password_hash) VALUES (%s, %s)",
                ("admin", hashed_password)
            )
            conn.commit()
            print("[+] Створено стандартного адміністратора (admin / admin). Змініть пароль у виробництві!")
        cursor.close()
        conn.close()
    except Exception as e:
```

```
        print(f"[!] Помилка ініціалізації адміна: {e}")


def require_api_key(func):
    def wrapper(*args, **kwargs):
        if request.headers.get('X-API-Key') == API_SECRET_KEY:
            return func(*args, **kwargs)
        else:
            abort(401, description="Unauthorized: Invalid API Key")
    wrapper.__name__ = func.__name__
    return wrapper


def login_required(func):
    def wrapper(*args, **kwargs):
        if not session.get('logged_in'):
            return redirect(url_for('login'))
        return func(*args, **kwargs)
    wrapper.__name__ = func.__name__
    return wrapper


@app.route('/')
@login_required
def admin_dashboard():
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM incidents ORDER BY timestamp DESC")
        incidents = cursor.fetchall()
        cursor.execute("SELECT * FROM policies LIMIT 1")
        policy = cursor.fetchone()
        cursor.close()
        conn.close()

        if not policy:
            policy = {"blocked_extensions": "", "forbidden_words": "", "regex_patterns": ""}

        return render_template('dashboard.html', incidents=incidents, policy=policy)
    except Exception as e:
        return f"Помилка завантаження: {e}"

@app.route('/update_policy', methods=['POST'])
@login_required
def update_policy():
    exts = request.form.get('extensions')
```

```
    words = request.form.get('words')
    regex = request.form.get('regex_patterns')

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "UPDATE policies SET blocked_extensions=%s, forbidden_words=%s, regex_patterns=%s WHERE id=1",
        (exts, words, regex)
    )
    conn.commit()
    cursor.close()
    conn.close()
    return redirect('/')


@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("SELECT * FROM admins WHERE username = %s", (username,))
        admin = cursor.fetchone()
        cursor.close()
        conn.close()


        if admin and check_password_hash(admin['password_hash'], password):
            session['logged_in'] = True
            session['username'] = admin['username']
            return redirect(url_for('admin_dashboard'))
        else:
            error = "Невірний логін або пароль!"

    return f'''
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <title>DLP Login</title>
        <style>
            body {{ font-family: sans-serif; display: flex; justify-content: center; align-items: center; height: 100vh; background-color: #f8f9fa; }}
```

```
                .login-box {{ background: white; padding: 40px; border-radius: 8px; box-shadow: 0 4px 12px rgba(0,0,0,0.1); text-align: center; width: 300px; }}
                input {{ padding: 10px; margin: 10px 0; width: 90%; border: 1px solid #ccc; border-radius: 4px; }}
                button {{ padding: 10px; background-color: #0d6efd; color: white; border: none; border-radius: 4px; cursor: pointer; width: 97%; font-weight: bold; }}
                .error {{ color: #dc3545; margin-bottom: 10px; font-size: 14px; }}
            </style>
        </head>
        <body>
            <div class="login-box">
                <h2>🛡️ DLP System</h2>
                <p>Панель керування безпекою</p>
                {f'<div class="error">{error}</div>' if error else ''}
                <form method="post">
                    <input type="text" name="username" placeholder="Логін..." required><br>
                    <input type="password" name="password" placeholder="Пароль..." required><br>
                    <button type="submit">Увійти</button>
                </form>
            </div>
        </body>
        </html>
        '''

    @app.route('/logout')
    def logout():
        session.pop('logged_in', None)
        session.pop('username', None)
        return redirect(url_for('login'))


    @app.route('/api/incidents', methods=['POST'])
    @require_api_key
    def receive_incident():
        data = request.json
        if not data: return jsonify({"error": "No data provided"}), 400
        try:
            conn = get_db_connection()
            cursor = conn.cursor()
            sql_insert = '''
                INSERT INTO incidents (timestamp, username, violation_type, details, action_taken)
                VALUES (%s, %s, %s, %s, %s)
            '''
            cursor.execute(sql_insert, (data['timestamp'], data['user'], data['violation_type'], data['details'], data['action_taken']))
            conn.commit()
            cursor.close()
            conn.close()
```

```
            return jsonify({"status": "success"}), 200
        except Exception as err:
            return jsonify({"status": "error", "message": str(err)}), 500

    @app.route('/api/policy', methods=['GET'])
    @require_api_key
    def get_policy():
        try:
            conn = get_db_connection()
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT blocked_extensions, forbidden_words, regex_patterns FROM policies LIMIT 1")
            policy = cursor.fetchone()
            cursor.close()
            conn.close()
            return jsonify(policy), 200
        except Exception as err:
            return jsonify({"error": str(err)}), 500

    if __name__ == '__main__':
        init_default_admin()

        print("=======================================================")
        print("[*] Enterprise DLP Server (Production Mode) Active")
        print("[*] Панель керування: http://127.0.0.1:5000")
        print("=======================================================")

serve(app, host='0.0.0.0', port=5000)
```

network_interceptor.py

```
    from mitmproxy import http
    import re
    import requests
    import io
    import datetime
    import threading
    import time
    from docx import Document
    from PyPDF2 import PdfReader

    API_SECRET_KEY = "SuperSecretEnterpriseKey2026"

    class DLPInterceptor:
        def __init__(self):

            self.forbidden_words = []
            self.regex_patterns = []
```

```
        self.blocked_extensions = []


        self.server_url = "http://127.0.0.1:5000"

        print("[*] Запуск фонової синхронізації політик з сервером...")
        threading.Thread(target=self.sync_policies, daemon=True).start()

    def sync_policies(self):
        """Завантажує нові правила з сервера кожні 10 секунд"""
        while True:
            try:
                response = requests.get(
                    f"{self.server_url}/api/policy",
                    headers={'X-API-Key': API_SECRET_KEY},
                    timeout=3,
                    proxies={"http": None, "https": None}
                )
                if response.status_code == 200:
                    data = response.json()


                    if 'forbidden_words' in data and data['forbidden_words']:
                        self.forbidden_words = [x.strip().lower() for x in data['forbidden_words'].split(',') if x.strip()]
                    else:
                        self.forbidden_words = []


                    if 'regex_patterns' in data and data['regex_patterns']:
                        raw_patterns = data['regex_patterns'].split('||')
                        compiled = []
                        for p in raw_patterns:
                            p = p.strip()
                            if p:
                                try: compiled.append(re.compile(p))
                                except: pass
                        self.regex_patterns = compiled
                    else:
                        self.regex_patterns = []


                    if 'blocked_extensions' in data and data['blocked_extensions']:
                        self.blocked_extensions = [x.strip().lower() for x in data['blocked_extensions'].split(',') if x.strip()]
                    else:
                        self.blocked_extensions = []
            except Exception:
```

```
            pass
        time.sleep(10)

    def send_alert(self, violation, details):
        if len(details) > 1000: details = details[:1000] + "... [ОБРІЗАНО]"
        payload = {
            "timestamp": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "user": "System (Network)",
            "violation_type": f"Мережевий витік ({violation})",
            "details": details,
            "action_taken": "Блокування (Пакет знищено)"
        }
        try:
            requests.post(
    f"{self.server_url}/api/incidents",
    json=payload,
    headers={'X-API-Key': API_SECRET_KEY},
    timeout=3,
    proxies={"http": None, "https": None}
)
            print("[+] Мережевий інцидент успішно записано в БД!")
        except Exception as e:
            print(f"[!] Помилка відправки логу на сервер: {e}")

    def analyze_text(self, text):
        text_lower = text.lower()
        for word in self.forbidden_words:
            if word and word in text_lower: return f"Секретне слово '{word}'"
        for pattern in self.regex_patterns:
            if pattern.search(text): return "Шаблонні дані (RegEx)"
        return None

    def analyze_file_in_memory(self, file_bytes, filename):

        ext = ""
        if '.' in filename:
            ext = f".{filename.split('.')[-1].lower()}"


        if ext and ext in self.blocked_extensions:
            return f"Заборонене розширення файлу ({ext})"


        extracted_text = ""
        try:
            if ext in ['.txt', '.csv', '.md']:
```

```
            extracted_text = file_bytes.decode('utf-8', 'ignore')
        elif ext == '.docx':
            doc = Document(io.BytesIO(file_bytes))
            extracted_text = "\n".join([p.text for p in doc.paragraphs])
        elif ext == '.pdf':
            reader = PdfReader(io.BytesIO(file_bytes))
            for page in reader.pages:
                extracted_text += (page.extract_text() or "")
    except Exception:
        pass

    return self.analyze_text(extracted_text)

def request(self, flow: http.HTTPFlow) -> None:

    if flow.request.host == "127.0.0.1" and flow.request.port == 5000:
        return

    if flow.request.method in ["POST", "PUT"]:
        content_type = flow.request.headers.get("Content-Type", "").lower()
        violation = None
        details = ""



        if "multipart/form-data" in content_type:
            if flow.request.multipart_form:
                for key, field in flow.request.multipart_form.items():
                    if hasattr(field, 'filename') and field.filename:
                        filename = field.filename.decode('utf-8', 'ignore')
                        violation = self.analyze_file_in_memory(field.content, filename)
                        if violation:
                            details = f"Файл: {filename} на {flow.request.host}"
                            break



        elif "application/json" in content_type or "application/x-www-form-urlencoded" in content_type:
            text_content = flow.request.content.decode('utf-8', 'ignore')
            violation = self.analyze_text(text_content)
            if violation:
                details = f"Текст на {flow.request.host}"



        elif len(flow.request.content) > 0:
            url_path = flow.request.path
            filename = url_path.split('/')[-1] if '/' in url_path else "unknown_file"
            violation = self.analyze_file_in_memory(flow.request.content, filename)
```

```
                if violation:
                    details = f"Сирий файл (Raw): {filename} на {flow.request.host}"

            if violation:
                print(f"🚨 МЕРЕЖЕВИЙ ІНЦИДЕНТ: {violation} -> {flow.request.host}")
                flow.response = http.Response.make(
                    403,
                    b"DLP Blocked: Data transfer contains confidential information.",
                    {"Content-Type": "text/html"}
                )
                self.send_alert(violation, details)

addons = [DLPInterceptor()]
```

dashboard.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DLP System - Admin Panel</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body { background-color: #f8f9fa; }
        .navbar { background-color: #0d1b2a; }
        .details-col { max-width: 200px; white-space: nowrap; overflow: hidden; text-overflow: ellipsis; }
    </style>
</head>
<body>

  <nav class="navbar navbar-dark mb-4">
      <div class="container-fluid">
          <span class="navbar-brand mb-0 h1"> DLP Security Dashboard</span>
          <div class="d-flex align-items-center">
              <span class="navbar-text text-light me-3">
                  Адміністратор: <strong>{{ session.username }}</strong>
              </span>
              <a href="/logout" class="btn btn-outline-danger btn-sm fw-bold"> Вийти з системи</a>
          </div>
      </div>
  </nav>

    <div class="container">
      <div class="container mt-4">
       <div class="card shadow-sm mb-4 border-primary">
```

```
                <div class="card-header bg-primary text-white">⚙ Налаштування політик безпеки (Синхронізація з агентами)</div>
                <div class="card-body">
                  <form action="/update_policy" method="POST">
                    <div class="row">
                      <div class="col-md-6 mb-3">
                        <label class="form-label fw-bold">Заборонені розширення файлів (через кому):</label>
                        <input type="text" class="form-control" name="extensions" value="{{ policy.blocked_extensions }}" placeholder=".exe, .bat, .mp3">
                        <small class="text-muted">Ці файли будуть миттєво видалятися при спробі запису на флешку.</small>
                      </div>
                      <div class="col-md-6 mb-3">
                        <label class="form-label fw-bold">Заборонені слова у вмісті (через кому):</label>
                        <input type="text" class="form-control" name="words" value="{{ policy.forbidden_words }}" placeholder="секрет, пароль">
                        <small class="text-muted">Будуть шукатися у буфері обміну та в текстових файлах на USB.</small>
                      </div>
                    </div>
                    <div class="row">
            <div class="col-12 mb-3">
              <label class="form-label fw-bold">Шаблони даних (Регулярні вирази, через розділювач ||):</label>
              <!-- Використовуємо || як розділювач, бо кома часто використовується всередині самих RegEx -->
              <input type="text" class="form-control text-monospace" name="regex_patterns" value="{{ policy.regex_patterns }}" placeholder="\b(?:\d[ -]*?){13,16}\b">
              <small class="text-muted">
                Приклади: \b\d{4}[ -]\d{4}[ -]\d{4}[ -]\d{4}\b (Банківські картки), <b>[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}</b> (Email).
              </small>
            </div>
          </div>
                    <button type="submit" class="btn btn-primary"> Зберегти та розіслати на комп'ютери</button>
                  </form>
                </div>
              </div>

              <div class="row mb-3">
                <div class="col">
                  <h3 class="text-secondary">Журнал інцидентів</h3>
                </div>
                <div class="col text-end">
                  <button onclick="window.location.reload();" class="btn btn-outline-secondary"> Оновити логи</button>
                </div>
              </div>


              <div class="card shadow-sm">
```

```
            <div class="card-body p-0">
                <table class="table table-hover table-striped mb-0">
                    <thead class="table-dark">
                        <tr>
                            <th>ID</th>
                            <th>Час інциденту</th>
                            <th>Користувач (ПК)</th>
                            <th>Тип загрози</th>
                            <th>Статус / Дія</th>
                            <th>Перехоплені дані</th>
                        </tr>
                    </thead>
                    <tbody>

                        {% for incident in incidents %}
                        <tr>
                            <td><strong>#{{ incident.id }}</strong></td>
                            <td>{{ incident.timestamp }}</td>
                            <td><span class="badge bg-secondary">{{ incident.username }}</span></td>
                            <td class="text-danger fw-bold">{{ incident.violation_type }}</td>
                            <td><span class="badge bg-danger">{{ incident.action_taken }}</span></td>
                            <td class="details-col" title="{{ incident.details }}">
                                <code>{{ incident.details }}</code>
                            </td>
                        </tr>
                        {% else %}

                        <tr>
                            <td colspan="6" class="text-center py-4 text-muted">
                                Інцидентів не виявлено. Система працює в штатному режимі.
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>

</body>
</html>
```

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Дмитро СЛОБОДЯНЮК

Співавтор:

Назва: Комп'ютерна система захисту конфіденційної інформації від витоків

Експерт: Олег САВЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:5.45%

Коефіцієнт подібності 2:1.99%

Мікропробіли: 3

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-06-03 01:44:44.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-06-03 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 19.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 273162 Назва: БКР Комп'ютерна система захисту конфіденційної інформації від витоків Додано в БД: 2026-06-02 Автора: Дмитро СЛОБОДЯНЮК Керівники: Олег САВЕНКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	97125	588	20265 (21%)	120 (20%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269488	Назва: Звіт з ПДП Комп'ютерна система захисту конфіденційної інформації від витоків Додано в БД: 2026-02-24 Автора: Слободянюк Д.О. Керівники: Лисенко С.М. Консультанти: Опоненти:	18779 (19.0%)	106 (18.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Слободянюк Дмитро Олександрович

Тема: Комп'ютерна система захисту конфіденційної інформації від витоків

Спеціальність**:** 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___59___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є проєктування та програмна реалізація розподіленої системи запобігання витокам інформації на основі багатопотокового аналізу кінцевих точок і мережевої інспекції для проактивного захисту даних

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи проведено аналіз предметної області захисту конфіденційної інформації та виявлено наявні проблеми й завдання в умовах розмиття корпоративного мережевого периметра і неконтрольованого використання штучного інтелекту. В другому розділі проведено проєктування децентралізованої трирівневої архітектури комп'ютерної системи захисту, обґрунтовано принципи розподілу обчислювального навантаження в умовах застосування парадигми периферійних обчислень та здійснено аналіз багатопотокової взаємодії фонових процесів агента на основі закону Амдала. В третьому розділі здійснено практичну програмну реалізацію спроєктованої комп'ютерної системи, розроблено алгоритми гарантованого затирання файлів, перехоплення клавіатури та інспекції мережевих потоків, спроєктовано веббазований інтерфейс для управління та проведено імітаційне тестування ефективності розробленого програмно-технічного комплексу.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: недостатня увага тестуванню програмного комплексу в умовах високого мережевого навантаження; недостатньо чітко описано процес розгортання агентів на кінцевих точках.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на достатньому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно (A / 90)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Терещенок Леонід Петрович, зав.каф.ІПЗ, ХНУ

"02" червня 2026 р. _____(підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Дмитро СЛОБОДЯНЮК

ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2С-23-2

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Комп'ютерна система захисту конфіденційної інформації від витоків
Автор Дмитро СЛОБОДЯНЮК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.т.н., професор Олег САВЕНКО

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.
4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 5,45%; та системою Anti-Plagiarism складає 1,99%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри _____ Підпис Ольга ПАВЛОВА Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис Андрій НІЧЕПОРУК Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис Олег САВЕНКО Ім'я, ПРІЗВИЩЕ