

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ
ІНФОРМАЦІЇ**

КАФЕДРА СИСТЕМ ТА ТЕХНОЛОГІЙ КІБЕРБЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

**«ТЕХНОЛОГІЯ КОРЕЛЯЦІЇ ТА ПРІОРИТИЗАЦІЇ ВРАЗЛИВОСТЕЙ НА
ОСНОВІ РЕЗУЛЬТАТІВ ВІДКРИТИХ СКАНЕРІВ»**

зі спеціальності

125 Кібербезпека та захист інформації

(код, найменування спеціальності)

освітньо-професійної програми

Інформаційна та кібернетична безпека

(назва програми)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Іван КОНДРАТЕНКО

(підпис)

Виконав: здобувач вищої освіти групи БСДМ-61

КОНДРАТЕНКО Іван

(прізвище, ім'я)

Керівник

д-р філос., доц., МАРЧЕНКО Віталій

(науковий ступінь, вчене звання, прізвище, ім'я)

Рецензент

_____ (науковий ступінь, вчене звання, прізвище, ім'я)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	3
ВСТУП.....	4
1.1. Поняття вразливостей та їх класифікація	6
1.2. Життєвий цикл управління вразливостями	11
1.3. Методи сканування.....	14
1.4. Проблеми традиційних підходів.....	17
1.5. Огляд наукових підходів до пріоритизації.....	20
2 МЕТОДИ ТА ЗАСОБИ ВИКОРИСТАННЯ ВІДКРИТИХ ІНСТРУМЕНТІВ СКАНУВАННЯ ВРАЗЛИВОСТЕЙ	26
2.1. Огляд існуючих безкоштовних рішень.....	26
2.2. Порівняння підходів до сканування та форматів результатів	31
2.3. Вибір інструментів для дослідження.....	38
2.4. Типові проблеми використання кількох сканерів одночасно	42
3 РОЗРОБЛЕННЯ ВАРІАНТА ТЕХНОЛОГІЇ КОРЕЛЯЦІЇ ТА ПРІОРИТИЗАЦІЇ ВРАЗЛИВОСТЕЙ.....	49
3.1. Архітектура запропонованої системи	49
3.2. Модель кореляції результатів	56
3.3. Власний підхід до пріоритизації.....	61
3.4. Програмна реалізація.....	69
ВИСНОВКИ	86
ПЕРЕЛІК ПОСИЛАНЬ.....	88
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація).....	90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	—	Application Programming Interface
ASPM	—	Application Security Posture Management
CI/CD	—	Continuous Integration / Continuous Deployment
CVE	—	Common Vulnerabilities and Exposures
CVSS	—	Common Vulnerability Scoring System
CWE	—	Common Weakness Enumeration
DAST	—	Dynamic Application Security Testing
DFD	—	Data Flow Diagram
FQDN	—	Fully Qualified Domain Name
GUI	—	Graphical User Interface
MTTR	—	Mean Time To Remediate
MVC	—	Model-View-Controller
NVD	—	National Vulnerability Database
OOP	—	Object-Oriented Programming
OS	—	Operating System
RBVM	—	Risk-Based Vulnerability Management
SAST	—	Static Application Security Testing
SCA	—	Software Composition Analysis
SIEM	—	Security Information and Event Management
SLA	—	Service Level Agreement
SOC	—	Security Operations Center
SQL	—	Structured Query Language
WAF	—	Web Application Firewall
XML	—	Extensible Markup Language
XSS	—	Cross-Site Scripting

ВСТУП

Актуальність дослідження. Стрімкий розвиток інформаційно-телекомунікаційних систем, широке впровадження технологій контейнеризації та мікросервісної архітектури призвели до експоненційного зростання поверхні атаки сучасних підприємств. За даними аналітичних звітів, кількість нових вразливостей (CVE), що реєструються щороку, перевищує десятки тисяч, що робить процес ручного управління ними неефективним. Існуючі комерційні рішення класу Vulnerability Management є високовартісними, тоді як доступні інструменти з відкритим кодом (Nmap, OWASP ZAP, Trivy) функціонують ізольовано, генеруючи розрізнені звіти у несумісних форматах. Це призводить до проблем фрагментації даних, масового дублювання результатів та інформаційного перевантаження персоналу («Alert Fatigue»), коли за великою кількістю технічних повідомлень втрачаються реальні загрози бізнесу. У зв'язку з цим, задача створення автоматизованої технології, здатної об'єднувати (корелювати) дані з різнорідних джерел та пріоритизувати їх на основі контексту, набуває особливої актуальності.

Об'єкт дослідження – процес управління вразливостями в інформаційних системах.

Предмет дослідження – методи, моделі та алгоритми автоматизованої кореляції, нормалізації та ризик-орієнтованої пріоритизації даних, отриманих від сканерів вразливостей з відкритим кодом.

Мета роботи – підвищення ефективності системи захисту інформації шляхом розробки технології, що забезпечує автоматизовану агрегацію результатів сканування, усунення дублювання даних та динамічну пріоритизацію інцидентів для зменшення часу реагування на критичні загрози.

Наукові завдання:

Проаналізувати сучасний стан проблеми управління вразливостями та існуючі інструментальні засоби, виявити їх недоліки.

Дослідити формати даних сканерів Nmap, OWASP ZAP, Trivy та розробити методи їх нормалізації.

Розробити алгоритми кореляції (дедуплікації) та мета-аналізу для виявлення складних векторів атак.

Обґрунтувати математичну модель адаптивної пріоритизації ризиків

Здійснити програмну реалізацію технології у вигляді автономного комплексу та перевірити його ефективність.

Методи дослідження – опрацювання методів системного аналізу – для дослідження предметної області та формалізації життєвого циклу вразливостей; теорію множин та реляційну алгебру – для розробки алгоритмів кореляції та структури бази даних; методи математичного моделювання – для побудови формули розрахунку пріоритетів; об'єктно-орієнтоване проектування – для розробки архітектури програмного засобу.

Практичне значення одержаних результатів полягає в розробці технології кореляції та пріоритизації вразливостей на основі результатів відкритих сканерів яка дозволяє автоматизувати рутинні процеси обробки результатів сканування, зменшити обсяг даних шляхом дедуплікації на 40-60% та забезпечити інтеграцію з корпоративними системами моніторингу (SIEM). Впровадження розробленого інструменту дозволяє організаціям перейти від реактивного усунення помилок до проактивного управління ризиками без значних капітальних витрат на пропрієтарне ПЗ.

Апробація результатів. Результати кваліфікаційної роботи апробовані на Всеукраїнській науково-практичній конференції «Актуальні проблеми кібербезпеки», яка відбулася 29 жовтня 2025 року в Державному університеті інформаційно-комунікаційних технологій, м. Київ.

1 АНАЛІЗ НЕОБХІДНОСТІ УПРАВЛІННЯ ВРАЗЛИВОСТЯМИ

1.1. Поняття вразливостей та їх класифікація

Розвиток інформаційно-телекомунікаційних систем (ІТС) в останнє десятиліття характеризується експоненційним зростанням складності архітектури програмного забезпечення та інфраструктури. Перехід від монолітних архітектур до мікросервісних, широке впровадження хмарних обчислень та технологій контейнеризації (Docker, Kubernetes) призвели до фундаментальної зміни парадигми забезпечення інформаційної безпеки. Якщо традиційний підхід базувався на концепції «фортеці» (захист периметра мережі), то сучасна доктрина, відображена в міжнародних стандартах NIST SP 800-137 та ISO/IEC 27001, фокусується на стратегії глибинного захисту (Defense in Depth) та безперервному моніторингу (Continuous Monitoring).

Ключовим елементом цієї нової парадигми є процес управління вразливостями (Vulnerability Management), об'єктом якого виступає поняття «вразливість» (vulnerability). В науково-технічному дискурсі існує низка дефініцій цього терміна, які варіюються залежно від контексту — від суто технічного до ризик-орієнтованого. Згідно з міжнародним стандартом ISO/IEC 27000:2018, вразливість визначається як «слабкість активу або управління, яка може бути використана однією або декількома загрозами». Це визначення є базовим, оскільки воно встановлює причинно-наслідковий зв'язок між трьома ключовими компонентами тріади безпеки: активом (ресурсом, що має цінність), самою вразливістю (внутрішньою слабкістю активу) та загрозою (зовнішнім або внутрішнім фактором, здатним експлуатувати вразливість) [1].

Національний інститут стандартів і технологій США (NIST) у спеціальній публікації SP 800-30 надає більш розширене та операційне трактування: «Вразливість — це недолік або слабкість у процедурах безпеки системи, проектуванні, впровадженні або внутрішньому контролі, які можуть бути

використані (випадково або навмисно) і призвести до порушення безпеки або порушення політики безпеки системи».

Для побудови математичних моделей оцінки захищеності, які використовуються в системах пріоритизації, важливо чітко розмежовувати поняття, що часто ототожнюються в популярній літературі. Вразливість (Vulnerability) є об'єктивною властивістю системи, наприклад, помилкою переповнення буфера у веб-сервері, і існує незалежно від того, чи є зловмисник, здатний її використати. Натомість загроза (Threat) — це потенційна подія або суб'єкт (наприклад, хакерське угруповання), здатний завдати шкоди. Інструментом реалізації загрози виступає експлойт (Exploit) - програмний код або послідовність команд. Кінцевою ж метрикою є ризик (Risk) - ймовірнісна міра, що характеризує можливість реалізації загрози через вразливість із нанесенням шкоди активу.

Розуміння природи вразливостей неможливе без аналізу життєвого циклу розробки програмного забезпечення (SDLC). Вразливості не є випадковими аномаліями; вони є закономірним результатом недосконалості технологічних процесів або людського фактору на різних етапах створення ІТС. Можна виділити три основні онтологічні класи вразливостей за походженням.

Першим і найбільш складним для виявлення класом є помилки проектування (Design Flaws). Вони закладаються на етапі архітектури системи. Автоматизовані сканери часто не здатні виявити їх напрямку, оскільки формально код працює без помилок, але сама логіка є вразливою. Прикладами є відсутність шифрування критичних даних при передачі між мікросервісами, використання слабких криптографічних алгоритмів (наприклад, MD5) або відсутність механізмів захисту від перебору (Rate Limiting). Їх детекція вимагає ручного аудиту або використання спеціалізованих DAST-сканерів, таких як OWASP ZAP.

Другим, найпоширенішим класом, є помилки реалізації (Implementation Bugs), що виникають на етапі написання коду. До них належать переповнення буфера (Buffer Overflow), некоректна санітизація вхідних даних (що призводить до SQL Injection або XSS) та стани гонитви (Race Condition). Такі вразливості

ефективно виявляються інструментами статичного аналізу (SAST) та аналізу залежностей.

Третій клас — це помилки конфігурації та експлуатації (Operational & Configuration Issues). Вони виникають не в коді програми, а в середовищі її виконання. Згідно зі звітами Verizon DBIR, неправильна конфігурація хмарних середовищ є однією з головних причин витоків даних. Типовими прикладами є використання паролів за замовчуванням, відкриті адміністративні інтерфейси або неправильні права доступу до файлів. Виявлення таких проблем є сферою відповідальності мережевих сканерів, зокрема Nmap.

Для побудови ефективної системи агрегації даних від різномірних сканерів необхідно застосувати багатовимірну класифікацію, яка дозволяє уніфікувати результати. Найбільш доцільним є використання моделі, що корелює рівень абстракції вразливості з інструментом її виявлення.

Вразливості мережевого та інфраструктурного рівня стосуються мережевого обладнання, операційних систем серверів та реалізації мережевих стеків протоколів TCP/IP. Сюди відносяться відкриті непотрібні порти, вразливі версії сервісів (SSH, FTP), слабкі параметри рукописання SSL/TLS. У розробленій системі за цей рівень відповідає сканер Nmap з модулем vulners, який аналізує банери сервісів та порівнює їх з базою CVE.

Вразливості прикладного рівня існують у коді веб-застосунків та API. Це найбільш динамічний вектор атак, що включає вразливості бізнес-логіки, клієнтські атаки (DOM-based XSS) та серверні ін'єкції. Для їх виявлення використовується OWASP ZAP, який працює методом DAST, емулюючи дії злоумисника.

Окремим пластом є вразливості середовища виконання та залежностей, притаманні контейнеризованим середовищам. Це вразливості у сторонніх бібліотеках, базових образах контейнерів або пакетах ОС. Для роботи з цим класом використовується інструмент Trivy, який реалізує метод SCA (Software Composition Analysis), скануючи файлову систему контейнера.

Важливо також враховувати специфіку методів сканування. Статичний аналіз (SAST) працює з вихідним кодом на етапі розробки і схильний до високого рівня хибних спрацьовувань. Динамічний аналіз (DAST), представлений OWASP ZAP та Nmap, працює з працюючим додатком ("чорна скринька") на етапі тестування або експлуатації. Сканування контейнерів (SCA), яке виконує Trivy, аналізує бінарні файли та менеджери пакетів, що забезпечує низький рівень хибних спрацьовувань завдяки детермінованому пошуку версій.

Однією з головних проблем при створенні систем кореляції є різноманітність форматів даних. Для вирішення цієї проблеми міжнародна спільнота розробила протокол автоматизації контенту безпеки (SCAP), ключовими компонентами якого є стандарти CVE, CWE та CVSS [2].

CVE (Common Vulnerabilities and Exposures) — це словник уніфікованих ідентифікаторів для публічно відомих вразливостей. Він виступає "первинним ключем" для об'єднання результатів Nmap та Trivy у єдину базу знань.

CWE (Common Weakness Enumeration) класифікує *типи* помилок програмного забезпечення, що дозволяє корелювати семантично схожі вразливості. Наприклад, різні сканери можуть по-різному називати XSS-атаку, але обидва посилатимуться на один і той самий ідентифікатор CWE-79. Це критично важливо для нормалізації даних від інструментів, що не використовують CVE (наприклад, ZAP).

CVSS (Common Vulnerability Scoring System) — це відкритий фреймворк для кількісної оцінки критичності вразливості. У роботі використовується версія CVSS v3.1. Вектор оцінки включає метрики експлуатації (вектор атаки, складність, необхідні права, участь користувача) та метрики впливу (конфіденційність, цілісність, доступність) [3].

Оскільки одним із компонентів системи є сканер OWASP ZAP, доцільно розглянути класифікацію вразливостей прикладного рівня згідно зі стандартом OWASP Top 10:2021. Цей документ відображає найбільш критичні ризики для веб-додатків, такі як порушення контролю доступу (Broken Access Control), криптографічні збої (Cryptographic Failures), ін'єкції (Injection) та небезпечний

дизайн (Insecure Design). Категорія "Вразливі та застарілі компоненти" (Vulnerable and Outdated Components) є точкою перетину результатів ZAP та Trivy, що підтверджує необхідність кореляції даних.

Критичним аспектом класифікації результатів сканування є їх достовірність, зокрема проблема хибних спрацювань (False Positives). У теорії виявлення сигналів це помилка першого роду, коли сканер звітує про загрозу, якої фактично не існує. Причинами можуть бути, наприклад, "бекпортування" патчів безпеки у старих версіях пакетів Linux, що вводять в оману сканери Nmap та Trivy, або особливості налаштування сторінок помилок веб-сервера, що плутають ZAP. У розробленій системі ця проблема вирішується шляхом впровадження механізму кореляції: гіпотеза полягає в тому, що ймовірність хибного спрацювання зменшується, якщо вразливість підтверджена кількома незалежними методами (наприклад, на рівні мережі та на рівні файлової системи контейнера).

Оскільки досліджувана система включає аналіз контейнерів, необхідно виділити специфіку вразливостей цього класу. Контейнерні образи мають шарувату структуру, де вразливості можуть знаходитися як у базовому образі (рівень ОС), так і в шарах залежностей додатку (рівень Application) або конфігурації. Ця специфіка вимагає від системи управління вразливостями здатності розрізняти вразливості, що можуть бути виправлені розробником, від тих, що потребують оновлення базової платформи.

Проведений аналіз теоретичних основ поняття «вразливість» дозволяє зробити висновок про багатовимірність цього об'єкта дослідження. Вразливість не може бути адекватно описана однією метрикою; необхідна комплексна модель, що враховує технічну тяжкість, семантичний тип, технологічний домен та контекст активу. Велика кількість джерел загроз вимагає інтеграції даних з різних доменів (Network, Web, Container), а недостатність "сирої" оцінки CVSS для прийняття управлінських рішень обґрунтовує необхідність врахування контексту активу та ймовірності хибного спрацювання. Використання стандартизованих ідентифікаторів CVE та метрик CVSS є фундаментом для програмної реалізації

алгоритмів автоматичної кореляції, що і буде реалізовано у практичній частині даної роботи.

1.2. Життєвий цикл управління вразливостями

Ефективність системи захисту інформації в сучасних та динамічних інфраструктурах не може бути забезпечена шляхом реалізації разових заходів або статичних конфігурацій безпеки. В умовах перманентної еволюції кіберзагроз, де час між виявленням нової вразливості (Zero-Day) та початком її масової експлуатації скорочується до годин, критичного значення набуває системний підхід до управління ризиками. Відповідно до рекомендацій спеціальної публікації Національного інституту стандартів і технологій США NIST SP 800-40r3 («Guide to Enterprise Patch Management Technologies») та міжнародного стандарту ISO/IEC 27002, управління вразливостями (Vulnerability Management, VM) розглядається не як одноразова дія, а як безперервний, циклічний процес ідентифікації, класифікації, пріоритизації, обробки та звітування про вразливості безпеки в інформаційних системах [6].

Перехід від реактивної моделі поведінки (реагування на інциденти постфактум) до проактивної (управління вразливостями до моменту їх експлуатації) вимагає чіткої формалізації життєвого циклу вразливості. Науково-технічна спільнота, зокрема інститути SANS та Gartner, визначає життєвий цикл управління вразливостями як замкнену ітеративну модель, що складається з взаємопов'язаних етапів, кожен з яких є необхідною умовою для успішного виконання наступного. Розрив цього ланцюга на будь-якому етапі призводить до критичного зниження загального рівня захищеності, створюючи ілюзію безпеки при фактичній відсутності контролю над поверхнею атаки.

Фундаментальним етапом, що ініціює цикл, є інвентаризація та виявлення активів (Asset Discovery). Аксиоматичним твердженням в теорії інформаційної безпеки є теза про неможливість захисту об'єкта, про існування або параметри якого адміністратору безпеки невідомо. У класичних статичних мережах цей етап

зводився до ведення реєстрів обладнання. Однак сучасні парадигми віртуалізації, хмарних обчислень та контейнеризації (зокрема, використання Docker та Kubernetes) радикально ускладнили це завдання. Активи в таких середовищах є ефемерними: час життя контейнера може вимірюватися хвилинами, а IP-адреси динамічно перерозподіляються [4]. Це породжує явище «тіньового IT» (Shadow IT) — появи несанкціонованих або не облікованих сервісів, які не потрапляють у периметр регулярного сканування та моніторингу, стаючи ідеальним плацдармом для атаки. Тому в рамках дослідження акцент робиться на автоматизованих методах розвідки: активному мережевому скануванню для виявлення "живих" хостів та аналізі реєстрів контейнерів для ідентифікації використовуваних образів програмного забезпечення.

Наступним логічним кроком є етап оцінки та детекції вразливостей (Vulnerability Assessment). На цьому етапі відбувається порівняння поточного стану виявлених активів з базами відомих сигнатур загроз (наприклад, базами CVE — Common Vulnerabilities and Exposures). Важливо зазначити, що ефективність детекції прямо залежить від обраного методу сканування. Як було визначено в попередніх розділах, для забезпечення повноти даних необхідно комбінувати різні підходи: агентний та безагентний, статичний (SAST) та динамічний (DAST). Безагентне сканування, яке реалізує інструмент Nmap, дозволяє оцінити систему з точки зору зовнішнього злоумисника («black box»), не впливаючи на конфігурацію цільового вузла. У той же час, інструменти аналізу складу програмного забезпечення (SCA), такі як Trivy, дозволяють виявляти вразливості на рівні залежностей та бібліотек ще до етапу розгортання, реалізуючи концепцію «Shift Left» — зміщення безпеки на ранні етапи розробки. Результатом цього етапу є масив «сирих» даних про вразливості, який у великих інфраструктурах може налічувати тисячі або десятки тисяч записів.

Саме тут виникає "пляшкове горлечко" традиційних систем управління вразливостями, що зумовлює необхідність виділення окремого етапу пріоритизації (Prioritization). Згідно зі статистичними даними аналітичних агенцій (зокрема Kenna Security та Gartner), середньостатистична організація має ресурси для

виправлення лише 5-10% від загальної кількості виявлених вразливостей. Ключовою науково-практичною задачею стає визначення саме тієї критичної меншості вразливостей, що становлять реальну, а не теоретичну загрозу. Традиційний підхід, що базується виключно на базовій оцінці CVSS (Base Score), довів свою недостатню ефективність в умовах сучасного ландшафту загроз. Вразливість з оцінкою CVSS 9.8 (Critical), що знаходиться на ізольованому тестовому сервері без доступу до Інтернету, становить значно менший ризик для бізнесу, ніж вразливість з оцінкою CVSS 7.5 (High) на публічному веб-сервері, що обробляє персональні дані клієнтів.

Сучасна концепція RBVM (Risk-Based Vulnerability Management), яка покладена в основу розробленої в даній роботі системи, пропонує перехід від одновимірної оцінки до багатофакторної моделі. Пріоритет вразливості розраховується як функція від технічної тяжкості, контексту активу (його критичності для бізнес-процесів) та достовірності виявлення. Саме на цьому етапі застосовуються технології кореляції даних від різних сканерів: якщо вразливість підтверджується незалежними інструментами (наприклад, мережевим сканером та аналізатором контейнерів), її пріоритет автоматично підвищується, оскільки це свідчить про високу ймовірність успішної експлуатації. Автоматизація цього процесу дозволяє значно скоротити час прийняття рішень (MTTD — Mean Time To Decide).

Четвертим етапом життєвого циклу є обробка та ремедіація (Remediation). На основі сформованого пріоритезованого списку приймається рішення щодо методу обробки ризику. Найбільш бажаним сценарієм є повне усунення вразливості (remediation), що зазвичай досягається шляхом встановлення патча (patching), оновлення версії програмного забезпечення або зміни конфігурації. Однак у реальних виробничих умовах це не завжди можливо через ризик порушення сумісності, відсутність патча від вендора (вразливості нульового дня) або неможливість зупинки критичних бізнес-процесів. У таких випадках застосовується стратегія мінімізації (mitigation) — впровадження компенсуючих заходів контролю, таких як налаштування правил міжмережевого екрану веб-

додатків (WAF), віртуальний патчінг або ізоляція вразливого сегмента мережі. Третім варіантом є прийняття ризику (risk acceptance), коли вартість усунення вразливості перевищує потенційні збитки від її експлуатації. Важливо, щоб таке рішення було свідомим, документованим та тимчасовим, із встановленням терміну перегляду.

Замикає цикл етап верифікації та моніторингу (Verification & Monitoring). Процес управління вразливостями не завершується встановленням оновлення. Необхідно провести повторне сканування (rescan), щоб підтвердити, що вразливість дійсно усунуто, і що процес виправлення не створив нових "дірок" у безпеці або не порушив функціональність системи. Крім того, цей етап включає безперервний моніторинг ландшафту загроз. Статус вразливості є динамічним: вразливість, яка вчора вважалася некритичною через відсутність публічного експлойту, сьогодні може стати вектором масованої атаки. Тому система повинна підтримувати актуальність бази знань та автоматично переглядати пріоритети при зміні зовнішніх факторів.

Аналіз життєвого циклу управління вразливостями дозволяє зробити висновок про наявність суттєвого технологічного розриву в існуючих інструментальних засобах, особливо у сегменті Open Source. Сучасні сканери ефективно автоматизують етапи виявлення та детекції, генеруючи великі масиви даних. Водночас етап пріоритизації часто залишається "ручним" процесом, що залежить від суб'єктивної оцінки експерта та не враховує контекст інфраструктури. Це призводить до збільшення часу середньої реакції на загрозу (MTTR — Mean Time To Remediate) та зниження загальної ефективності системи захисту. Розробка технології автоматизованої кореляції та пріоритизації, що є предметом даного дослідження, спрямована саме на вирішення цієї проблеми, забезпечуючи інтеграцію розрізнених даних у єдину керовану систему прийняття рішень.

1.3. Методи сканування

Вирішення науково-технічної задачі побудови ефективної системи управління вразливостями неможливе без детального аналізу існуючих методів отримання первинних даних про стан захищеності інформаційної системи. У сучасній теорії інформаційної безпеки під терміном «сканування вразливостей» (vulnerability scanning) розуміють автоматизований процес виявлення слабких місць у програмному забезпеченні, апаратному комплексі або мережевій конфігурації шляхом порівняння параметрів об'єкта дослідження з базою даних відомих сигнатур загроз. Згідно з методологією PTES (Penetration Testing Execution Standard) та стандартом OSSTMM (Open Source Security Testing Methodology Manual), вибір методу сканування є визначальним фактором, що впливає на повноту (completeness) та достовірність (accuracy) отриманих результатів, які надалі стають вхідними даними для підсистеми кореляції [15].

На основі аналізу наукової літератури та практичних підходів до аудиту безпеки можна виділити три основні групи методів, які є релевантними для даного дослідження: активне мережеве сканування, динамічний аналіз застосунків (DAST) та аналіз складу програмного забезпечення (SCA).

Фундаментальним методом, що лежить в основі будь-якого процесу аудиту, є активне мережеве сканування. Сутність цього методу полягає у відправленні спеціально сформованих пакетів (проб) на цільові вузли мережі та аналізі отриманих відповідей. Цей підхід дозволяє визначити топологію мережі, доступність хостів, перелік відкритих портів та версії запущених служб (Service Fingerprinting). У контексті даної роботи базовим інструментом для реалізації цього методу обрано Nmap (Network Mapper). Наукова спільнота визнає Nmap стандартом де-факто завдяки його гнучкості у використанні різних технік сканування, таких як TCP SYN scan (напіввідкрите сканування, що дозволяє залишатися непоміченим для простих систем логування), UDP scan та OS detection (визначення операційної системи за особливостями реалізації стека TCP/IP). Критичною перевагою Nmap для задач даного дослідження є наявність скриптового рушія NSE (Nmap Scripting Engine), який дозволяє не лише виявляти порти, але й автоматично співставляти банери сервісів з базою даних вразливостей

CVE. Це дозволяє отримати первинний вектор даних про вразливості інфраструктурного рівня, що є необхідним компонентом для подальшої кореляції.

Другим критично важливим методом є динамічний аналіз безпеки застосунків (DAST — Dynamic Application Security Testing). На відміну від мережевого сканування, яке працює на рівнях L3-L4 моделі OSI, DAST оперує на прикладному рівні (L7), розглядаючи цільову систему як «чорну скриньку». Метод полягає в емуляції дій зовнішнього зловмисника, який взаємодіє з веб-інтерфейсом або API додатку. Сканер DAST автоматично формує HTTP/HTTPS запити, впроваджуючи в них вектори атак (payloads) для виявлення вразливостей, таких як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та некоректна конфігурація заголовків безпеки. Для цілей даної роботи як інструмент DAST обрано OWASP ZAP (Zed Attack Proxy). Вибір цього інструменту обґрунтований його архітектурою, що дозволяє працювати в режимі проксі-сервера, перехоплюючи та модифікуючи трафік, а також наявністю потужного API, що дозволяє автоматизувати процес сканування та інтегрувати його в розроблювану систему кореляції. Важливість DAST у цьому дослідженні зумовлена тим, що значна частина критичних вразливостей сучасних систем знаходиться саме на рівні бізнес-логіки веб-застосунків, і вони не можуть бути виявлені традиційними мережевими сканерами.

Третім методом, що набуває особливої актуальності в умовах переходу до хмарних технологій та мікросервісної архітектури, є аналіз складу програмного забезпечення (SCA — Software Composition Analysis) та сканування контейнерів. Специфіка сучасного програмного забезпечення полягає в тому, що до 80-90% коду кінцевого продукту складають сторонні бібліотеки (Open Source dependencies) та базові образи операційних систем. Традиційні методи (мережеве сканування та DAST) часто не здатні ефективно виявити вразливості всередині "замороженого" образу контейнера, який ще не запуснений у середовищі виконання. Метод SCA передбачає статичний аналіз файлової системи, маніфестів пакетних менеджерів (наприклад, package.json, requirements.txt) та конфігураційних файлів для виявлення компонентів з відомими вразливостями. У рамках роботи для реалізації цього методу використовується інструмент Trivy. Його перевагою є висока

швидкість роботи, підтримка широкого спектру операційних систем та мов програмування, а також здатність виявляти вразливості не лише в прикладному ПЗ, але й у системних бібліотеках OS-рівня. Включення методу SCA до загальної схеми дослідження дозволяє покрити клас загроз Supply Chain Attacks, які є невидимими для зовнішнього спостерігача.

Аналіз наукових джерел та практичного досвіду вказує на те, що жоден із зазначених методів, застосований ізольовано, не забезпечує достатньої повноти покриття поверхні атаки. Мережеве сканування може пропустити вразливості веб-додатку на відкритому порті 80; DAST не бачить застарілих бібліотек на рівні ОС сервера; SCA не може оцінити, чи є вразлива бібліотека реально доступною для експлуатації з мережі. Саме тому виникає об'єктивна необхідність у комплексному підході, що поєднує результати різнорідних методів сканування. Така гібридна модель сканування, яку пропонується реалізувати в даній кваліфікаційній роботі, створює передумови для ефективної кореляції даних. Наприклад, якщо мережевий сканер (Nmap) виявляє відкритий порт, а сканер контейнерів (Trivy) підтверджує наявність вразливого сервісу, що слухає цей порт, достовірність такої знахідки значно зростає. І навпаки, якщо SCA-сканер знаходить вразливість у бібліотеці, яка не використовується жодним активним сервісом, виявленням DAST-сканером, пріоритет такої вразливості може бути знижений. Таким чином, інструментальний базис дослідження сформовано на основі трьох комплементарних методів: активного мережевого зондування (Nmap), динамічного аналізу веб-застосунків (OWASP ZAP) та статичного аналізу вмісту контейнерів (Trivy). Такий вибір дозволяє охопити всі рівні функціонування сучасної інформаційної системи — від мережевої інфраструктури до коду додатку та його залежностей, що забезпечує необхідну повноту вхідних даних для розробки технології кореляції та пріоритизації вразливостей.

1.4. Проблеми традиційних підходів

Аналіз методів сканування та життєвого циклу вразливостей, проведений у попередніх підрозділах, дозволяє констатувати, що сучасний ринок засобів інформаційної безпеки насичений потужними інструментами для детекції загроз. Використання рішень з відкритим кодом, таких як Nmap, OWASP ZAP та Trivy, дозволяє з високою точністю ідентифікувати технічні недоліки на різних рівнях інфраструктури. Однак, парадоксальним наслідком впровадження цих інструментів у виробниче середовище стає не підвищення рівня захищеності, а виникнення кризи управління даними. Традиційний підхід, що базується на ізольованому використанні спеціалізованих сканерів без єдиної системи оркестрації та аналітики, стикається з низкою системних проблем, які унеможливають ефективне реагування на інциденти в умовах дефіциту часу та людських ресурсів.

Фундаментальною проблемою є фрагментація даних (Data Silos). Кожен із розглянутих інструментів оперує у своєму власному контексті та семантичному полі. Мережевий сканер Nmap "бачить" інфраструктуру як набір IP-адрес та портів; веб-сканер ZAP оперує URL-адресами та HTTP-запитами; сканер контейнерів Trivy працює з іменами пакетів та хешами образів. У традиційній моделі результати їх роботи представлені у вигляді розрізнених звітів (XML, JSON, HTML), які не мають перехресних посилань. Адміністратор безпеки змушений вручну співставляти, що вразливість CVE-2021-44228 (Log4Shell), виявлена Nmap на порту 8080, та вразлива бібліотека log4j-core-2.14.1.jar, знайдена Trivy у файловій системі контейнера, — це одна й та сама сутність. Відсутність автоматизованої нормалізації даних призводить до дублювання зусиль та втрати цілісної картини поверхні атаки.

Другою критичною проблемою є явище, відоме в англomовній літературі як «Alert Fatigue» (втома від тривог) або інформаційне перевантаження. Сучасні сканери налаштовані на максимальну чутливість, щоб уникнути пропуску загроз (мінімізація False Negatives), що неминуче призводить до генерації величезної кількості повідомлень про вразливості низького рівня критичності або інформаційного характеру. Згідно зі звітами аналітичної компанії Kenna Security,

середній обсяг вразливостей у корпоративних мережах сягає мільйонів, при цьому лише 2-5% з них реально експлуатуються. В умовах ручного аналізу відділи безпеки фізично не здатні обробити такий потік інформації, що призводить до ігнорування звітів та пропуску реальних критичних загроз серед "білого шуму".

Третім аспектом є відсутність бізнес-контексту при пріоритизації. Традиційні інструменти покладаються виключно на базову оцінку CVSS (Base Score), яка є статичною метрикою, визначеною вендором ПЗ або дослідником у момент виявлення вразливості. Ця оцінка не враховує специфіку конкретного середовища експлуатації. Наприклад, вразливість з CVSS 9.0 (Critical) на сервері розробки, який не має доступу до Інтернету і не містить конфіденційних даних, становить значно менший ризик для організації, ніж вразливість з CVSS 6.0 (Medium) на платіжному шлюзі. Традиційні підходи не мають вбудованих механізмів для врахування цінності активу (Asset Criticality) та топології мережі, що призводить до неефективного розподілу ресурсів: ІТ-спеціалісти витрачають час на встановлення патчів на некритичних системах, залишаючи вразливими ключові бізнес-сервіси.

Четвертою проблемою є високий рівень хибних спрацювань (False Positives) при ізольованому скануванні. Як зазначалося в попередніх розділах, кожен метод сканування має свої сліпі зони. Мережевий сканер може помилково ідентифікувати версію сервісу за банером, який був змінений адміністратором для заплутування злоумисників (Security through Obscurity). Статичний аналізатор може знайти вразливу функцію у бібліотеці, яка фактично ніколи не викликається у працюючому додатку ("мертвий код"). У традиційному підході верифікація кожного такого спрацювання лягає на плечі оператора, перетворюючи процес управління вразливостями на рутинну перевірку гіпотез замість реального усунення загроз. Відсутність механізму крос-валідації (коли дані одного сканера підтверджують або спростовують дані іншого) є суттєвим недоліком існуючих безкоштовних рішень.

Нарешті, суттєвою проблемою є часова затримка (Time Lag) між виявленням та виправленням. У ручному режимі процес збору звітів, їх консолідації в Excel-таблиці, узгодження з власниками систем та формування завдань для

адміністраторів може займати тижні або навіть місяці. У цей період (Time to Remediate) інфраструктура залишається відкритою для атак. В умовах, коли експлойти для нових вразливостей з'являються у публічному доступі протягом 48-72 годин після розкриття інформації (як це було у випадку з Exim, Exchange, Log4j), така затримка є неприпустимою.

Підсумовуючи аналіз проблематики, можна стверджувати, що традиційна модель управління вразливостями, побудована на використанні розрізнених інструментів без єдиної аналітичної надбудови, вичерпала свій потенціал. Вона характеризується високою трудомісткістю, низькою оперативністю та відсутністю фокусу на реальних ризиках. Це обґрунтовує нагальну необхідність розробки автоматизованої системи класу Vulnerability Correlation & Prioritization, яка здатна програмно вирішити проблеми нормалізації даних, фільтрації шуму та контекстної пріоритизації, що є головною метою даної кваліфікаційної роботи.

1.5. Огляд наукових підходів до пріоритизації

Критичний аналіз проблематики управління вразливостями, проведений у попередніх підрозділах, засвідчив, що ключовим викликом для сучасних систем інформаційної безпеки є не дефіцит даних про загрози, а неспроможність ефективно їх обробляти. Ця ситуація стимулювала активний розвиток наукових досліджень у напрямку розробки методів та алгоритмів пріоритизації (Vulnerability Prioritization). В науковій літературі та стандартах індустрії сформувалося кілька концептуальних підходів до вирішення цієї задачі, які еволюціонували від статичних метрик до динамічних адаптивних моделей.

Історично першим і досі найбільш поширеним підходом є використання стандартизованих скорингових систем, еталоном яких є CVSS (Common Vulnerability Scoring System), що розробляється організацією FIRST.org. Науковий базис CVSS полягає у декомпозиції вразливості на вектор метрик. Проте, як зазначають дослідники Scarfone та Mell у роботах NIST, CVSS Base Score за своєю природою відображає лише «технічну тяжкість» (technical severity), а не реальний

ризик [8]. Формула CVSS є детермінованою і статичною: вразливість отримує оцінку в момент публікації і ця оцінка рідко змінюється, навіть якщо з'являється експлойт або змінюється ландшафт загроз. Дослідження Allodi та Massacci показали, що базування стратегії виправлення виключно на оцінках CVSS > 7.0 є неефективним, оскільки значна частина вразливостей з високим рейтингом ніколи не експлуатується в реальних атаках (wild), тоді як деякі вразливості з середнім рейтингом стають частиною складних ланцюжків атак (kill chains). Незважаючи на введення у версіях CVSS v3.1 та v4.0 темпоральних (Temporal) та контекстних (Environmental) метрик, на практиці організації рідко їх використовують через складність ручного розрахунку для тисяч активів.

У відповідь на обмеження статичних моделей виник напрямок імовірнісного прогнозування експлуатації. Яскравим представником цього підходу є система EPSS (Exploit Prediction Scoring System), яка розвивається під егідою FIRST. Наукова новизна EPSS полягає у зміні парадигми з оцінки тяжкості («наскільки погана ця вразливість?») на оцінку загрози («яка ймовірність того, що цю вразливість спробують експлуатувати у найближчі 30 днів?»). Модель EPSS використовує методи машинного навчання (Machine Learning), навчаючись на масивах даних про реальні кібератаки. Фактори моделі включають:

- Наявність публічного експлойту (PoC) або коду в Metasploit/ExploitDB.
- Вік вразливості та популярність програмного продукту.
- Згадки в соціальних мережах, блогах безпеки та Dark Web.
- Характеристики вектора атаки (віддалена/локальна).

EPSS генерує ймовірність від 0 до 1 (0% - 100%), що оновлюється щоденно. Дослідження показують, що стратегія пріоритизації на основі EPSS є значно ефективнішою за CVSS з точки зору зменшення реальної поверхні атаки при менших витратах ресурсів (виправлення меншої кількості вразливостей дає більший ефект зниження ризику). Однак, EPSS має суттєвий недолік для локальних інфраструктур: вона оцінює глобальну загрозу ("threat landscape"), але абсолютно ігнорує внутрішній контекст ("asset context"). Наприклад, сервер, захищений WAF

та ізолюваний від Інтернету, матиме ту ж оцінку EPSS, що й публічний сервер, що є методологічною проблемою [5].

Третім, якісно відмінним напрямком, є контекстно-орієнтовані моделі прийняття рішень, найбільш відомою з яких є методологія SSVC (Stakeholder-Specific Vulnerability Categorization), розроблена Університетом Карнегі-Меллона (SEI CMU) спільно з Агентством кібербезпеки та безпеки інфраструктури США (CISA). SSVC відмовляється від лінійного численного ранжування (наприклад, 1-10) на користь дерев прийняття рішень (Decision Trees). Це дозволяє уникнути "магії чисел", коли різниця між 7.8 і 7.9 є несуттєвою, але може впливати на черговість робіт [14].

Алгоритм SSVC базується на проходженні вразливості через дерево рішень з чотирма вузлами:

1. Exploitation (Статус експлуатації): Чи є докази активної експлуатації? (None, PoC, Active).
2. Technical Impact (Технічний вплив): Частковий контроль або повний контроль над системою?
3. Automatable (Автоматизованість): Чи можуть зловмисники автоматизувати атаку (черв'яки)?
4. Mission Impact (Вплив на місію): Наскільки критичним є актив для організації?

Результатом проходження дерева є одна з чотирьох директив дії:

- Track: Моніторити ситуацію, не вживати активних дій.
- Attend: Звернути увагу, виправити у плановому порядку.
- Act: Виправити якомога швидше (стандартний SLA).
- Act Immediately: Негайна реакція (Incident Response).

SSVC є методологічно найближчим до реальних задач управління, проте його автоматизація вимагає наявності точної та актуальної бази даних активів (Asset Inventory) з розміченими рівнями критичності, що часто відсутнє у невеликих організаціях або в динамічних хмарних середовищах.

Окремий кластер досліджень присвячений використанню графів атак (Attack Graphs). Цей метод, описаний у фундаментальних роботах Sheyner, Jha та Wing, моделює інфраструктуру як орієнтований граф, де вузли представляють можливі стани системи (наприклад, "права користувача на сервері А", "права root на сервері Б"), а ребра — можливі переходи між станами через експлуатацію вразливостей. Пріоритизація у такій моделі базується на аналізі шляхів досягнення компрометації критичних активів ("корони"). Вразливість вважається пріоритетною, якщо вона знаходиться на критичному шляху (critical path) або є точкою перетину багатьох шляхів атак (cut set), усунення якої розриває ланцюжок атаки. Хоча цей підхід є найбільш точним з точки зору моделювання загроз, він характеризується високою обчислювальною складністю (проблема пошуку мінімального розрізу часто є NP-hard) та вимагає надзвичайно детальної та актуальної моделі мережевої топології та прав доступу, що ускладнює його впровадження у реальному часі.

Також варто відзначити підходи, що базуються на кореляції даних (Data Fusion). У сучасних роботах (Wang, Guo) пропонується використовувати методи сенсорного злиття для об'єднання результатів різнорідних джерел безпеки. Ідея полягає у тому, що перетин множин даних від різних інструментів (наприклад, IDS, сканерів вразливостей, логів додатків) дозволяє значно підвищити достовірність (Confidence) виявлення проблеми. Саме цей напрямок є найбільш перспективним для практичної реалізації в умовах обмежених ресурсів. Він дозволяє компенсувати недоліки окремих сканерів (наприклад, високий рівень False Positive у DAST-інструментів) за рахунок перехресної перевірки даними з іншого домену (наприклад, SAST або Container Scanning), не вимагаючи при цьому побудови надскладних математичних моделей графів атак.

Аналіз розглянутих підходів дозволяє зробити висновок, що на сьогодні не існує універсальної "срібної кулі". Комерційні рішення класу VM (Tenable, Qualys, Rapid7) інтегрують елементи CVSS, Threat Intelligence (аналог EPSS) та частково Attack Graphs, але вони є закритими "чорними скриньками" з високою вартістю ліцензування. У сегменті Open Source рішень спостерігається дефіцит інструментів,

які б реалізували гібридну модель пріоритизації, що поєднувала б технічну оцінку (CVSS), контекст активу (як у SSVC) та кореляційний коефіцієнт достовірності.

Отже, існує об'єктивна науково-практична потреба у розробці технології, яка б адаптувала передові наукові підходи (зокрема, зважування контексту активу та кореляцію даних) для використання з доступними інструментами (Nmap, ZAP, Trivy). Така технологія повинна заповнити нішу між простими агрегаторами звітів та складними комерційними системами аналітики, надаючи можливість автоматизованої пріоритизації на основі прозорої математичної моделі ризику, що і є метою даної кваліфікаційної роботи.

Висновки до першого розділу

У першому розділі кваліфікаційної роботи проведено комплексний аналіз теоретичних та методологічних засад управління вразливостями в сучасних інформаційних системах.

1. Розкрито сутність досліджуваного явища: Вразливість визначено як багатовимірний об'єкт, управління яким вимагає системного підходу, що включає етапи виявлення, оцінки, пріоритизації та усунення. Показано, що в умовах динамічних середовищ (контейнеризація, мікросервіси) статичні методи захисту втрачають ефективність.

2. Проаналізовано інструментарій: Обґрунтовано вибір комплексу відкритих інструментів (Nmap, OWASP ZAP, Trivy), який забезпечує повне покриття рівнів інфраструктури (мережевий, прикладний, рівень залежностей). Визначено, що ізольоване використання цих інструментів призводить до фрагментації даних та інформаційного перевантаження.

3. Ідентифіковано проблему: Встановлено, що ключовим бар'єром ефективності є відсутність у доступних інструментах механізмів автоматизованої пріоритизації, яка б враховувала бізнес-контекст активів та кореляцію результатів. Традиційні підходи, що базуються лише на CVSS, генерують надмірну кількість хибних пріоритетів.

4. Визначено напрямок досліджень: Огляд наукових підходів підтвердив перспективність методів кореляції даних та ризик-орієнтованої пріоритизації. Це формує теоретичне підґрунтя для розробки власної технології у наступних розділах, яка має об'єднати результати різнорідних сканерів у єдину метрику ризику для прийняття обґрунтованих управлінських рішень.

2 МЕТОДИ ТА ЗАСОБИ ВИКОРИСТАННЯ ВІДКРИТИХ ІНСТРУМЕНТІВ СКАНУВАННЯ ВРАЗЛИВОСТЕЙ

2.1. Огляд існуючих безкоштовних рішень

У першому розділі кваліфікаційної роботи було визначено теоретичні засади управління вразливостями та обґрунтовано необхідність автоматизації процесів кореляції та пріоритизації. Другий розділ присвячено аналізу практичних аспектів реалізації цих процесів. Метою даного підрозділу є комплексне дослідження існуючого ландшафту програмних засобів з відкритим кодом (Open Source), що призначені для агрегації даних сканування, а також виявлення їх архітектурних та функціональних обмежень, які зумовлюють необхідність розробки власного спеціалізованого рішення.

Ринок систем управління вразливостями (Vulnerability Management Systems — VMS) історично характеризується глибокою поляризацією. З одного боку, існують потужні комерційні платформи корпоративного рівня (Enterprise), такі як Tenable.sc, Rapid7 InsightVM або Qualys VMDR. Ці системи пропонують повний цикл автоматизації, включаючи сканування, пріоритизацію на основі Threat Intelligence та патч-менеджмент. Однак, їх висока вартість ліцензування, закритість коду («black box») та орієнтація на великі корпоративні мережі роблять їх недоступними або надлишковими для використання у межах даного дослідження, а також для потреб малого та середнього бізнесу. З іншого боку, існує сегмент рішень з відкритим кодом (Open Source), які розвиваються спільнотою ентузіастів та експертів з безпеки. Саме цей сегмент є предметом детального аналізу, оскільки він дозволяє дослідити алгоритми обробки даних «зсередини» та виявити методичні прогалини, які потребують вирішення.

Аналіз репозиторіїв GitHub, профільних ресурсів (OWASP, ToolsWatch) та наукових публікацій дозволив виділити найбільш репрезентативні платформи, що реалізують функції агрегації даних від різномірних сканерів. До них належать

OWASP DefectDojo, Faraday IDE та ArcherySec. Кожна з цих систем представляє окремий підхід до архітектури та методології обробки вразливостей.

Безумовним лідером та стандартом де-факто у світі Open Source оркестрації безпеки є платформа OWASP DefectDojo. Це комплексний інструмент класу ASPM (Application Security Posture Management), написаний мовою Python з використанням фреймворку Django. DefectDojo позиціонується як «єдине вікно» для команд DevSecOps, дозволяючи імпортувати звіти з понад 150 різних інструментів безпеки, включаючи Nmap, OWASP ZAP, Trivy, Burp Suite, SonarQube та багато інших. Архітектурно DefectDojo є складною розподіленою системою, що включає веб-сервер (uWSGI/Nginx), реляційну базу даних (PostgreSQL або MySQL), брокер повідомлень (RabbitMQ або Redis) для асинхронної обробки завдань та воркери (Celery) [12].

Функціональна модель DefectDojo базується на ієрархії об'єктів «*Product Type → Product → Engagement → Test → Finding*». Така структура дозволяє детально моделювати процеси розробки, прив'язуючи вразливості до конкретних білдів або релізів програмного забезпечення. Однією з найсильніших сторін DefectDojo є вбудований механізм дедуплікації (Deduplication). Система використовує алгоритми порівняння хешів, згенерованих на основі ключових полів вразливості (наприклад, CVE ID, URL, номер рядка коду), щоб автоматично об'єднувати ідентичні знахідки, виявлені під час повторних сканувань. Це частково вирішує проблему «інформаційного шуму», описану в першому розділі. Крім того, платформа підтримує двосторонню інтеграцію з системами відстеження помилок (Jira), що дозволяє автоматично створювати тікети для розробників при виявленні нових вразливостей.

Однак, при всій потужності функціоналу, DefectDojo має низку суттєвих недоліків у контексті задачі побудови легкої та адаптивної системи пріоритизації. По-перше, це архітектурна надлишковість та високі вимоги до ресурсів. Розгортання повноцінного інстансу DefectDojo вимагає оркестрації щонайменше 5-6 контейнерів, а мінімальні вимоги до оперативної пам'яті складають 8–16 ГБ. Це робить систему «важкою» для локального використання на робочій станції

аудитора або на периферійних серверах. По-друге, незважаючи на можливості дедуплікації, DefectDojo слабо реалізує саме кореляцію між різними типами сканерів. Система чудово справляється з агрегацією (збором) звітів, але не має вбудованої логіки для автоматичного підвищення пріоритету вразливості на основі перехресного підтвердження. Наприклад, якщо Nmap виявив відкритий порт, а Trivy — вразливий сервіс на цьому порту, DefectDojo відобразить це як дві окремі знахідки в рамках різних «Тестів», покладаючи задачу їх логічного зв'язування на оператора. По-третє, модель пріоритизації в DefectDojo є переважно статичною. Хоча система дозволяє вручну змінювати рівень критичності (Severity) та додавати теги, вона не має автоматизованого двигуна (Calculation Engine), який би перераховував фінальний бал ризику на основі ваг активів та поточного ландшафту загроз у реальному часі [9].

Іншим популярним підходом є використання інструменту Faraday IDE (Integrated Penetration-Test Environment). На відміну від DefectDojo, який орієнтований на довгострокове управління життєвим циклом вразливостей (Vulnerability Management), Faraday розроблений передусім для команд пентестерів (Red Teams) та проведення активних операцій з оцінки захищеності. Його філософія базується на інтеграції з терміналом користувача: спеціальний демон перехоплює команди, що вводяться в консолі (наприклад, запуск nmap або sqlmap), парсить їх вивід у реальному часі та заносить результати до бази даних. Це дозволяє створювати єдиний простір для спільної роботи команди аудиторів.

Архітектурно Faraday використовує клієнт-серверну модель, де серверна частина (базується на Python/Flask та PostgreSQL) відповідає за зберігання та обробку даних, а клієнтська частина (GTK-інтерфейс або Web UI) надає інструменти візуалізації. Сильною стороною Faraday є візуалізація поверхні атаки та можливість побудови графів зв'язків між хостами та сервісами. Проте, з точки зору задач даного дослідження, Faraday має критичні обмеження. По-перше, його функціонал «заточений» під ручну верифікацію та експлуатацію вразливостей, а не під автоматизовану пріоритизацію великих масивів даних. Система не пропонує алгоритмів розрахунку ризику, покладаючись на експертну оцінку пентестера. По-

друге, значна частина аналітичних функцій (наприклад, розширені звіти, порівняння сканувань, планувальник завдань) винесена у платну версію (Pro/Corp), що обмежує можливості використання Community-версії для побудови повноцінної системи управління вразливостями без додаткових витрат. Крім того, Faraday фокусується на мережевій безпеці та веб-додатках, маючи обмежену підтримку специфіки контейнеризованих середовищ та вразливостей ланцюга постачання (SCA), які є важливою частиною сучасного ландшафту загроз.

Третім розглянутим рішенням є ArcherySec. Це проект з відкритим кодом, який позиціонується як спрощена альтернатива DefectDojo. ArcherySec дозволяє керувати сканерами (OpenVAS, ZAP, Burp) та імпортувати їх результати. Особливістю даного інструменту було широке використання стеку ELK (Elasticsearch, Logstash, Kibana) для зберігання та візуалізації даних у ранніх версіях, що забезпечувало потужні можливості пошуку та фільтрації. Однак, аналіз динаміки розвитку проекту показує суттєве зниження активності розробників в останні роки (стагнація). Багато бібліотек та залежностей проекту є застарілими, що створює ризики безпеки при впровадженні самого інструменту захисту (парадокс "сапожника без чобіт"). Функціонально ArcherySec працює переважно як агрегатор звітів, надаючи зручні дашборди, але не пропонуючи глибокої аналітичної обробки даних. Механізми кореляції в ньому рудиментарні і зводяться до простого групування за типом вразливості, без врахування топології мережі чи критичності активів.

Окремо варто згадати клас інструментів, що являють собою прості парсери або конвертери форматів (наприклад, Dradis, Seccubus). Dradis Framework є потужним засобом для генерації звітів, дозволяючи об'єднувати виводи різних інструментів у єдиний документ Word або PDF. Проте Dradis фокусується саме на етапі звітування (Reporting), а не на аналізі. Він не зберігає історію змін стану вразливості (Lifecycle Management) і не надає інструментів для математичного розрахунку пріоритетів. Seccubus, у свою чергу, спеціалізується на дельта-аналізі (пошуку різниці між скануваннями), що є корисною функцією, але недостатньою для комплексного управління ризиками.

Узагальнюючи аналіз існуючих рішень, можна виділити системну проблему ринку Open Source засобів управління вразливостями: розрив між агрегацією та кореляцією. Більшість існуючих платформ (DefectDojo, ArcherySec) чудово справляються із задачею агрегації — тобто збором різнорідних даних в одному місці. Вони вирішують проблему нормалізації форматів (про що йтиметься у наступному підрозділі), дозволяючи відобразити XML від Nmap та JSON від Trivy в єдиній таблиці. Однак, вони не виконують кореляцію у глибокому, семантичному сенсі цього слова. Кореляція передбачає встановлення логічних зв'язків між подіями. Наприклад, якщо мережевий сканер бачить порт 80 відкритим, а веб-сканер знаходить на ньому SQL-ін'єкцію, а сканер контейнерів бачить вразливу версію бібліотеки libpq (PostgreSQL client) у цьому сервісі — це все частини одного пазлу, які мають бути об'єднані в один «Meta-Finding» з підвищеним пріоритетом. Існуючі системи відображають це як три незалежні проблеми, змушуючи аналітика витрачати час на їх співставлення.

Другою системною проблемою є статичність моделей пріоритизації. Розглянуті платформи за замовчуванням імпортують оцінку CVSS зі звіту сканера. Якщо Nmap каже, що вразливість має 7.5 балів, DefectDojo відобразить 7.5. Вони не мають вбудованого гнучкого двигуна, який би дозволив користувачу задати формулу: «Якщо актив має тег "Production" і вразливість підтверджена двома сканерами, то помножити CVSS на 1.2; якщо актив "Test" — помножити на 0.5». Саме така контекстна адаптація є ключем до вирішення проблеми перевантаження алертами (Alert Fatigue).

Таким чином, проведений огляд дозволяє зробити висновок про наявність незаповненої ніші у сегменті інструментарію. Існує об'єктивна потреба у розробці легкого (Lightweight), автономного програмного рішення, яке б поєднувало можливості універсального парсингу (як у DefectDojo), але фокусувалося б саме на математичній кореляції результатів та динамічній пріоритизації на основі контексту активів. Така система має бути менш ресурсомною, ніж Enterprise-платформи, та більш аналітично глибокою, ніж прості агрегатори звітів. Це обґрунтовує доцільність розробки власного програмного продукту в рамках даної

кваліфікаційної роботи, архітектура та алгоритми якого будуть детально описані у третьому розділі.

2.2. Порівняння підходів до сканування та форматів результатів

У попередньому підрозділі кваліфікаційної роботи було здійснено аналіз існуючих програмних рішень для управління вразливостями, що дозволило виявити їхні архітектурні обмеження, зокрема в контексті гнучкості обробки даних та адаптивності до специфічних потреб організації. Наступним логічним та критично важливим етапом дослідження є структурно-семантична декомпозиція та порівняльний аналіз безпосередньо первинних даних, що генеруються сканерами безпеки. Глибоке розуміння синтаксичної та семантичної структури звітів (Report Structures) є фундаментальною передумовою для розробки ефективних алгоритмів їх автоматизованої обробки (Parsing), нормалізації (Normalization) та подальшої кореляції.

Відсутність єдиного стандарту представлення результатів сканування є однією з головних проблем сучасної індустрії інформаційної безпеки. Хоча існують спроби стандартизації (наприклад, формат SARIF — Static Analysis Results Interchange Format), більшість популярних інструментів продовжують використовувати власні пропріетарні схеми даних. У рамках даної роботи як джерела вхідних даних обрано тріаду інструментів з відкритим кодом, що забезпечують покриття різних рівнів архітектури сучасної інформаційної системи:

1. Nmap (Network Mapper) — для аудиту мережевого рівня, транспортних протоколів та інфраструктурних сервісів.
2. OWASP ZAP (Zed Attack Proxy) — для динамічного аналізу веб-додатків (DAST) та виявлення вразливостей прикладного рівня.
3. Trivy — для статичного аналізу контейнеризованих середовищ, образів Docker та ланцюга постачання програмного забезпечення (SCA).

Нижче наведено детальний технічний аналіз форматів даних кожного з цих інструментів, виявлені проблеми інтероперабельності та наукове обґрунтування

підходів до їх вирішення, що були покладені в основу програмної реалізації системи кореляції.

Інструмент Nmap є визнаним стандартом де-факто для мережевого аудиту та інвентаризації активів. Хоча Nmap підтримує різні формати виводу (Interactive, Greppable), для задач машинної обробки та інтеграції найбільш придатним є формат XML (Extensible Markup Language), структура якого суворо регламентується DTD-схемою nmaprun. Аналіз згенерованих звітів дозволив виділити ієрархічну модель даних, яка фокусується на сутностях «Хост» (Host) та «Порт» (Port).

Кореневим елементом будь-якого звіту Nmap є тег `<nmaprun>`, який містить критично важливі метадані про контекст сканування: версію сканера (scanner), повний рядок аргументів командного рядка (args), час початку сканування у форматі Unix Timestamp (start) та версію XML-виводу. Наприклад, аналіз атрибуту args дозволяє програмно верифікувати, чи використовувалися під час сканування скрипти NSE (`-sC` або `--script`), що є індикатором потенційної наявності даних про вразливості у звіті.

Основною інформаційною одиницею звіту є блок `<host>`. Цей елемент агрегує всю інформацію про окремий мережевий вузол. Критично важливими дочірніми елементами є:

- `<status>`: визначає поточний стан хоста (`state="up"` або `state="down"`) та причину такого визначення (reason, наприклад, `arp-response` або `syn-ack`). Це поле є першим фільтром у розробленому парсері, що дозволяє відсіювати неактивні вузли ще до етапу аналізу портів;
- `<address>`: ідентифікує ціль сканування. Важливою технічною особливістю є те, що один хост може мати декілька елементів `<address>` з різними типами (addrtype): `ipv4`, `ipv6` та `mac`. Це створює проблему однозначної ідентифікації активу. У розробленому алгоритмі (parsers.py) реалізовано пріоритет вибору адреси: IPv4 є первинним ключем, тоді як MAC-адреса використовується як додатковий атрибут для кореляції в локальних мережах;
- `<ports>`: контейнер, що містить перелік усіх просканованих портів;

- `<port>`: описує конкретний порт, його номер (portid) та протокол (protocol). Вкладений елемент `<service>` надає інформацію про виявлений програмний продукт (product), його версію (version) та, що важливо для кореляції, тип операційної системи (ostype), визначений методом OS Fingerprinting.

Найбільш складним та проблематичним аспектом парсингу Nmap є те, що сам по собі він є сканером портів, а не вразливостей. Функцію детекції вразливостей (CVE) виконує його розширюваний скриптовий рушій NSE (Nmap Scripting Engine), зокрема популярний скрипт vulners. Детальний аналіз XML-структури показує, що результати роботи будь-яких NSE-скриптів розміщуються у загальному контейнері `<script>`, вкладеному в тег `<port>` (якщо скрипт стосується сервісу) або `<host>` (якщо скрипт стосується ОС) [7].

Критична проблема полягає у форматі представлення цих даних. Інформація про знайдені вразливості, включаючи їх ідентифікатори (CVE ID) та оцінки критичності (CVSS Score), передається не як структуровані XML-теги (наприклад, `<cve id="..." score="...">`), а як неструктурований текстовий блок (Plain Text) всередині атрибуту output.

Це створює фундаментальну проблему для автоматизації: стандартні XML-парсери (такі як xml.etree у Python) здатні вилучити лише весь текст атрибуту цілком, але не можуть розрізнити окремі вразливості всередині нього. Для вирішення цієї науково-технічної задачі у розробленому модулі системи було реалізовано гібридний підхід до обробки даних. На першому етапі використовується SAX або DOM парсер для навігації по дереву XML до потрібного порту. На другому етапі до вмісту атрибуту output застосовуються методи контент-аналізу на основі регулярних виразів (Regular Expressions). Розроблений алгоритм ідентифікує патерни типу `CVE-\d{4}-\d{4,7}` та пов'язані з ними числа з плаваючою комою (CVSS Score), що дозволяє структурувати "сирий" вивід та перетворити його на об'єкти вразливостей, придатні для запису в базу даних. Така реалізація, хоч і є обчислювально складнішою, забезпечує можливість отримання даних про вразливості з інструменту, який первинно для цього не був оптимізований.

Сканер веб-застосунків OWASP ZAP функціонує за методологією DAST, розглядаючи цільову систему як "чорну скриньку" та взаємодіючи з нею виключно через протокол HTTP/HTTPS. Це кардинально змінює модель даних звіту порівняно з інфраструктурними сканерами. ZAP підтримує вивід у форматах XML та JSON. У рамках роботи було проаналізовано XML-структуру як найбільш стабільну та деталізовану версію звіту [10].

Основним інформаційним елементом звіту ZAP є об'єкт `<alertitem>` (у новіших версіях може бути згрупований у контейнер `<site>`). Цей об'єкт описує конкретний тип вразливості (Alert), виявлений на веб-ресурсі. Ключові атрибути, що підлягають екстракції:

- `<pluginid>`: унікальний числовий ідентифікатор плагіна ZAP, що виявив проблему. Цей ID є стабільним і може використовуватися для дедуплікації;
- `<alert>`: назва вразливості, зрозуміла для людини (наприклад, "X-Content-Type-Options Header Missing" або "SQL Injection");
- `<riskcode>` та `<riskdesc>`: рівень ризику, визначений сканером;
- `<uri>`: конкретна URL-адреса, на якій спрацював експлойт;
- `<param>`: параметр HTTP-запиту (GET/POST змінна), що виявився вразливим до ін'єкції;
- `<evidence>`: фрагмент відповіді сервера, що підтверджує наявність вразливості (Proof of Concept).

Критичною проблемою інтеграції даних ZAP у єдину систему пріоритизації є використання ним власної, дискретної шкали оцінки ризиків, яка є семантично не сумісною з безперервною шкалою CVSS (0.0–10.0), що використовується Nmap та Trivy. ZAP використовує поле `riskcode` зі значеннями:

- 0: Informational (Інформаційний);
- 1: Low (Низький);
- 2: Medium (Середній);
- 3: High (Високий).

Це створює "семантичний розрив" (semantic gap). Неможливо напряму математично порівняти вразливість від Nmap з CVSS 9.8 та вразливість від ZAP з

RiskCode 3. Для вирішення цієї проблеми у розробленій системі (модуль `parsers.py`) реалізовано механізм нормалізації метрик (Score Mapping). Впроваджено логіку конвертації, яка присвоює умовні бали CVSS відповідним рівням ZAP: High ≥ 8.0 (що відповідає нижній межі діапазону High у CVSS v3), Medium ≥ 5.5 , Low ≥ 2.5 , Info ≥ 0.0 . Такий підхід є евристичним наближенням, проте він є необхідним компромісом для приведення різнорідних метрик до єдиного знаменника, що дозволяє розраховувати загальний пріоритет за єдиною формулою [11].

Ще однією особливістю ZAP є те, що він рідко надає ідентифікатори CVE, оскільки помилки веб-додатків (наприклад, XSS або Insecure Deserialization) часто є специфічними для кастомного коду, а не для версії відомого продукту. Натомість ZAP широко використовує класифікатори CWE (Common Weakness Enumeration) та WASC ID. Це вимагає, щоб база даних розробленої системи (`vuln_db.sqlite`) підтримувала зберігання не лише CVE, але й CWE, що було враховано при проектуванні схеми БД.

Також аналіз реальних звітів ZAP виявив проблему надмірної гранулярності: сканер генерує окремий запис `<alertitem>` для кожної URL-адреси, де знайдено одну й ту ж вразливість. Наприклад, відсутність заголовка безпеки на 100 сторінках сайту призведе до 100 записів у звіті, що засмічує базу даних. Для боротьби з цим у системі реалізовано механізм агрегації екземплярів (Instance Aggregation): під час імпорту система перевіряє, чи вже існує запис про такий тип вразливості (`pluginid`) для даного хоста. Якщо так — нова URL-адреса додається як "екземпляр" (Instance) до існуючого запису батьківської вразливості, замість створення дубліката.

Інструмент Trivy, призначений для SCA (Software Composition Analysis), представляє третій клас джерел даних і використовує формат JSON (JavaScript Object Notation), що є стандартом для сучасних Cloud Native інструментів. Структура JSON-звіту Trivy є об'єктно-орієнтованою і значно складнішою за "плоскі" списки Nmap чи ZAP.

Звіт Trivy складається з кореневого масиву об'єктів Results. Кожен об'єкт відповідає певній цілі сканування (Target). Це принципово важлива архітектурна відмінність: один образ контейнера може містити декілька цілей, наприклад:

1. Операційна система (Target: "alpine:3.14 (alpine 3.14.2)") — містить вразливості системних пакетів (apk, apt, rpm), класифіковані як Class: "os-pkgs".
2. Файли залежностей (Target: "app/package-lock.json") — містить вразливості бібліотек мов програмування (npm, pip, maven), класифіковані як Class: "lang-pkgs".

Всередині кожного об'єкта Result знаходиться масив Vulnerabilities. Об'єкт вразливості у Trivy характеризується надзвичайною деталізацією:

- VulnerabilityID: Основний ідентифікатор (CVE-XXXX-XXXX або CWE-XXXX-XXXX).
- PkgName: Назва вразливого пакету.
- InstalledVersion: Поточна версія пакету.
- FixedVersion: Версія, у якій вразливість виправлено. Наявність цього поля дозволяє системі автоматично генерувати рекомендації типу "Upgrade [PkgName] to [FixedVersion]".
- Layer: Інформація про шар файлової системи Docker (Digest та DiffID), у якому знайдено вразливий файл.

Унікальною проблемою при парсингу Trivy, виявленою в ході дослідження, є наявність декількох оцінок CVSS для однієї і тієї ж вразливості від різних джерел. Об'єкт CVSS у JSON-звіті є словником, що може містити дані від різних вендорів: nvd, redhat, ubuntu, debian.

Оцінки можуть суттєво відрізнятися через різницю в методології оцінювання (наприклад, Red Hat часто знижує оцінку, якщо вразливий код не компілюється у стандартній збірці RHEL). Для розробленої системи це означає необхідність реалізації алгоритму пріоритизації джерел (Source Selection Algorithm). У функції parse_trivy_json реалізовано каскадну логіку, яка спочатку намагається отримати оцінку від nvd (як найбільш універсальну метрику від NIST), а за її відсутності — від вендора (redhat або іншого), що забезпечує максимальну консистентність оцінок у межах системи.

Проведений декомпозиційний аналіз форматів даних дозволяє зробити висновок про наявність великої кількості різноманітних форматів, що

унеможливилює використання простих методів агрегації (конкатенації) звітів. Виявлені розбіжності можна класифікувати за трьома вимірами:

1. Синтаксична несумісність (Syntactic Heterogeneity): Інструменти використовують різні формати серіалізації даних (XML проти JSON) та радикально різні схеми іменування полів. Якщо Nmap оперує атрибутами XML (addr, portid), то Trivy використовує ключі JSON (Target, VulnerabilityID). Це вимагає розробки окремих програмних адаптерів (парсерів) для кожного інструменту, які б реалізовували патерн проектування "Adapter" або "Facade", приводячи вхідні дані до єдиного внутрішнього представлення [16].

2. Семантична несумісність (Semantic Heterogeneity): Найбільш критичним аспектом є відсутність єдиної метрики вимірювання ризику.

- Nmap: Надає "сирий" CVSS Score, часто схований у тексті.
- ZAP: Надає якісні рівні (Low/High), що вимагають мапінгу.
- Trivy: Надає складні об'єкти з множинними векторами CVSS від різних вендорів. Без попередньої нормалізації (приведення до єдиної шкали) неможливо побудувати коректний список пріоритетів. Система просто не зможе вирішити алгоритмічно, що є більш пріоритетним: "High" від ZAP чи "7.8" від Nmap.

3. Адресна несумісність та проблема ідентифікації активів (Asset Identification): Кожен сканер "бачить" актив через призму своєї технології:

- Для Nmap активом є IP-адреса.
- Для ZAP активом є URL-адреса.
- Для Trivy активом є назва образу або шлях до файлу.

У реальній інфраструктурі всі ці сутності можуть належати до однієї логічної системи (наприклад, веб-сервер на IP 192.168.1.10 обслуговує URL <http://myapp.local> та запущений у контейнері myapp:latest). Відсутність наскрізного ідентифікатора (Global ID) у звітах сканерів робить задачу кореляції нетривіальною. Це обґрунтовує необхідність створення власної бази даних активів (Asset Inventory), яка б виконувала роль сполучної ланки (Mapping Table), дозволяючи пов'язати результати сканування IP-адреси з результатами сканування контейнера.

Таким чином, проведений аналіз диктує жорсткі архітектурні вимоги до розроблюваної системи кореляції: необхідність модульної підсистеми парсингу для кожного типу сканера, впровадження канонічної моделі даних ("Normalized Finding") з уніфікованими полями, та потребу в "розумній" кореляції, здатній встановлювати логічні зв'язки між різними типами ідентифікаторів активів. Тільки вирішення проблеми нормалізації даних дозволить перейти до наступного етапу — розробки математичних моделей пріоритизації, які базуватимуться на довірених та порівнюваних метриках ризику.

2.3. Вибір інструментів для дослідження

Процес проектування та програмної реалізації системи автоматизованої кореляції та пріоритизації вразливостей вимагає комплексного підходу до формування інструментального базису. На основі аналізу, проведеного у попередніх підрозділах, для отримання первинних даних про стан захищеності обрано комплекс сканерів з відкритим кодом: Nmap (для мережевої інфраструктури), OWASP ZAP (для веб-додатків) та Trivy (для контейнеризованих середовищ). Однак, для побудови єдиної системи управління цими інструментами, агрегації їх результатів та реалізації алгоритмів кореляції необхідний відповідний технологічний стек розробки. Вибір засобів реалізації (мови програмування, СКБД, бібліотек) здійснено на основі критеріїв швидкодії обробки текстових даних, надійності зберігання інформації та простоти розгортання.

В якості базової мови програмування для реалізації системи обрано Python 3. Цей вибір не є випадковим і зумовлений домінуючою позицією Python у сфері інформаційної безпеки (InfoSec) та Data Science. Порівняно з компільованими мовами, такими як C++ або Java, Python як інтерпретована мова високого рівня забезпечує значно вищу швидкість розробки прототипів (Rapid Prototyping) завдяки лаконічному синтаксису та динамічній типізації. Хоча компільовані мови потенційно пропонують вищу продуктивність обчислень, у контексті даної роботи "вузьким місцем" (bottleneck) є не процесорний час, а операції введення-виведення

(I/O) — читання файлів звітів та мережева взаємодія. Для таких задач накладні витрати інтерпретатора Python є нехтуваними порівняно зі зручністю маніпулювання текстовими даними. Крім того, Python має одну з найбагатших стандартних бібліотек ("batteries included"), що дозволяє вирішувати складні задачі без залучення важких сторонніх фреймворків, що зменшує поверхню атаки самого інструменту. Зокрема, критично важливою для проекту стала наявність потужних вбудованих засобів для роботи зі структурованими даними: бібліотеки `xml.etree.ElementTree` для парсингу ієрархічних XML-структур (використовується для звітів Nmap та OWASP ZAP) та модуля `json` для роботи з об'єктною моделлю звітів Trivy.

Окремого обґрунтування вимагає вибір архітектури взаємодії з обраними зовнішніми сканерами. Оскільки розроблена система функціонує як оркестратор, вона повинна керувати процесами запуску консольних утиліт (Nmap, Docker-контейнери). Для цієї мети в середовищі Python ідеально підходить модуль `subprocess`, який надає повний контроль над потоками введення-виведення (STDIN/STDOUT/STDERR) дочірніх процесів. Це дозволяє не лише запускати сканування, але й перехоплювати вивід у реальному часі для відображення логів у графічному інтерфейсі, а також коректно обробляти коди завершення процесів для визначення статусу виконання завдання. Альтернативні підходи, такі як використання специфічних API-обгорток (wrappers), були відхилені через їхню потенційну застарілість або несумісність з новими версіями сканерів, тоді як прямий виклик через `subprocess` гарантує сумісність з будь-якою версією інструменту, встановленого в системі.

Для організації збереження даних (Data Persistence Level) було проведено аналіз між використанням повнофункціональних клієнт-серверних систем управління базами даних (таких як PostgreSQL або MySQL) та вбудованих (embedded) рішень. Вибір було зроблено на користь реляційної СКБД SQLite. Це рішення продиктоване архітектурною концепцією "легкого" (lightweight) та автономного інструменту. Використання PostgreSQL вимагало б від користувача розгортання та адміністрування окремого серверного процесу, налаштування

мережевих доступів та аутентифікації, що значно підвищує поріг входження та ускладнює дистрибуцію програмного продукту. Натомість SQLite є безсерверною бібліотекою, яка інтегрується безпосередньо в процес додатку, а вся база даних зберігається у вигляді єдиного кросплатформеного файлу на диску.

Незважаючи на свою компактність, SQLite забезпечує повну відповідність принципам ACID (Atomicity, Consistency, Isolation, Durability), що гарантує цілісність даних навіть у випадку аварійного завершення роботи програми або збою живлення. Це критично важливо для системи безпеки, де втрата результатів багатогодинного сканування є неприпустимою. Підтримка повноцінної мови SQL дозволила реалізувати складну реляційну модель даних, що включає таблиці для активів, груп, історії сканувань та знайдених вразливостей. Можливість використання складних JOIN-запитів та індексів забезпечує високу швидкість вибірки даних для кореляції та фільтрації, що було б складно реалізувати при використанні NoSQL рішень (наприклад, MongoDB) або примітивному зберіганні даних у файлах CSV/JSON [13].

Ключовим компонентом взаємодії з користувачем є графічний інтерфейс (GUI). У сучасній розробці існує тенденція до переходу на веб-інтерфейси (Electron, Django, React), проте для задач даного дослідження було обрано бібліотеку Tkinter — стандартний GUI-фреймворк для Python, що є обгорткою над бібліотекою Tk. Цей вибір ґрунтується на принципах безпеки та мінімалізму. Веб-інтерфейси, навіть локальні, вимагають запуску веб-сервера, що автоматично відкриває нові вектори атак (XSS, CSRF, Injection, перехоплення трафіку на локальному інтерфейсі). Використання нативного десктопного GUI усуває цей клас загроз: програма працює в ізольованому віконному середовищі операційної системи з правами поточного користувача. Крім того, Tkinter входить до стандартної дистрибуції Python, що забезпечує "Zero Dependency" розгортання — користувачеві не потрібно встановлювати Node.js або компілювати складні залежності [17]. Для забезпечення чуйності інтерфейсу (Responsiveness) при виконанні тривалих операцій сканування у додатку реалізовано багатопотокову архітектуру з використанням модуля threading та безпечних поточкових черг queue.

Це дозволяє виконувати блокуючі I/O операції (мережеве сканування) у фонових потоках, залишаючи головний потік інтерфейсу вільним для обробки подій користувача, що є стандартом якості для сучасних десктопних додатків.

Для реалізації аналітичних та звітних функцій системи до технологічного стеку було включено спеціалізовані бібліотеки. Для візуалізації статистичних даних (розподіл вразливостей за критичністю, статусами тощо) використано бібліотеку Matplotlib. Її потужний API дозволяє програмно генерувати високоякісні графіки та діаграми. Важливою особливістю є можливість інтеграції Matplotlib безпосередньо у вікна Tkinter через механізм FigureCanvasTkAgg, що дозволяє створювати інтерактивні дашборди без необхідності збереження проміжних зображень на диск [19].

Для етапу документування результатів (Reporting) обрано бібліотеку ReportLab, яка є промисловим стандартом для генерації PDF-документів у середовищі Python. На відміну від бібліотек, що конвертують HTML у PDF (як wkhtmltopdf), ReportLab дозволяє формувати документ на низькому рівні, малюючи текст, лінії та таблиці безпосередньо на "полотні" (Canvas). Це забезпечує повний контроль над версткою, точне позиціонування елементів та можливість створення професійних звітів зі складною структурою, підтримкою кирилических шрифтів (Unicode) та векторною графікою.

Додатково, для забезпечення інтеграції з корпоративними системами моніторингу подій безпеки (SIEM — Security Information and Event Management), у проекті використано бібліотеку CEFEvent. Вона дозволяє стандартизувати вихідні повідомлення системи у форматі CEF (Common Event Format), який підтримується більшістю сучасних SIEM-рішень (HP ArcSight, IBM QRadar, Splunk) [18]. Це перетворює розроблений інструмент з локального сканера на повноцінний елемент екосистеми безпеки підприємства.

Підсумовуючи, можна стверджувати, що обраний технологічний стек (Python + SQLite + Tkinter + ReportLab) є результатом компромісу між функціональною потужністю та простотою експлуатації. Відмова від складної клієнт-серверної архітектури на користь монолітного десктопного додатку з вбудованою БД

дозволила створити автономний, надійний та безпечний інструмент, який може бути розгорнутий на будь-якій робочій станції аналітика за лічені хвилини. Такий підхід відповідає сучасним вимогам до інструментів класу "Security Automation Tooling", де цінується швидкість адаптації, відсутність жорстких інфраструктурних вимог та можливість глибокої кастомізації логіки роботи.

2.4. Типові проблеми використання кількох сканерів одночасно

Проведений у попередніх підрозділах структурно-семантичний аналіз інструментального базису та форматів даних дозволив виявити суттєві архітектурні розбіжності у моделях представлення результатів сканування. На сучасному етапі розвитку технологій інформаційної безпеки комплексне використання кількох спеціалізованих інструментів, таких як Nmap, OWASP ZAP та Trivy, є не просто рекомендацією, а необхідною умовою забезпечення повноти покриття поверхні атаки ("Attack Surface Coverage"). Це зумовлено тим, що жоден окремий інструмент не володіє достатньою "шириною погляду" для виявлення всього спектру загроз у середовищі: мережевий сканер залишається "сліпим" до вразливостей бібліотек усередині контейнерів, SAST-інструмент не здатен виявити проблеми динамічної конфігурації веб-сервера, а DAST-сканер не бачить інфраструктурних вад операційної системи. Однак, ізольоване застосування цих різнорідних засобів без наявності єдиної системи оркестрації та аналітики породжує складний комплекс системних проблем, які парадоксальним чином можуть знижувати загальну ефективність процесу управління вразливостями, перетворюючи надлишок даних на дефіцит корисної інформації. Аналіз практичного досвіду експлуатації таких систем та результатів експериментальних досліджень дозволяє стверджувати, що ключовими бар'єрами є масове дублювання та фрагментація даних, критичне інформаційне перевантаження персоналу, відоме як "Alert Fatigue", висока ймовірність хибних спрацювань та проблема часової розсинхронізації результатів.

Однією з найбільш фундаментальних та очевидних проблем є масове дублювання та семантична неузгодженість результатів сканування. Оскільки

функціональні можливості сучасних сканерів частково перетинаються, один і той самий технічний недолік часто виявляється різними інструментами, але описується у різних термінах, контекстах та з використанням різних ідентифікаторів. Розглянемо класичний приклад критичної вразливості у криптографічній бібліотеці OpenSSL (наприклад, CVE-2014-0160, відома як "Heartbleed"). Мережевий сканер Nmap, скануючи порт 443, за допомогою модуля ssl-heartbleed виявить вразливість через аналіз відповіді сервера на спеціально сформований пакет рукоштовування (handshake) і відобразить це у звіті як проблему мережевого сервісу, прив'язану до конкретної IP-адреси. Паралельно з цим, сканер контейнерів Trivy, аналізуючи файлову систему Docker-образу, на якому базується цей веб-сервер, знайде застарілий бінарний пакет libssl1.0.0 і згенерує запис про ту ж саму CVE, але прив'язану до назви пакету та хешу шару образу. У той же час, сканер OWASP ZAP, виконуючи активне сканування веб-додатку, може виявити витік пам'яті сервера або специфічні HTTP-заголовки, що свідчать про використання вразливої версії SSL/TLS, і створити алерт "Vulnerable OpenSSL Version Disclosed".

У результаті такої мультивекторної детекції адміністратор безпеки отримує три окремі повідомлення про одну й ту ж проблему. У "сирому", ненормалізованому вигляді ці дані сприймаються як три різні інциденти: один стосується мережевої інфраструктури, другий — розробки та збірки контейнерів (DevOps), третій — безпеки веб-додатків (AppSec). У ручному режимі роботи це вимагає від аналітика значних інтелектуальних зусиль та витрат часу для ментальної кореляції цих фактів. Фахівець повинен вручну співставити IP-адресу сервера з контейнером, що на ньому запущений, та URL-адресою, яку він обслуговує, щоб дійти висновку, що це не три незалежні загрози, а одна сутність, яка підтверджена з трьох різних точок зору. Відсутність автоматизованої дедуплікації призводить до штучного роздування статистики вразливостей (іноді на 200-300%), що спотворює метрики безпеки та призводить до неефективного розподілу ресурсів, коли різні команди (мережеві адміністратори та розробники) намагаються виправити одну й ту ж проблему паралельно, часто конфліктуючи між собою. Окрім дублювання, ситуація ускладнюється семантичною неузгодженістю:

різні сканери використовують різні назви для одних і тих же типів вразливостей (наприклад, "HTTP Trace Method Enabled" у Nmap проти "Cross-Site Tracing" у ZAP), що змушує аналітика постійно звертатися до зовнішніх баз знань (CWE, CAPEC) для верифікації синонімічності знахідок.

Другою, не менш критичною проблемою, що виникає при одночасному використанні кількох інструментів, є явище, відоме в науковій літературі як "Alert Fatigue" (втома від тривоги). Сучасні сканери за замовчуванням налаштовані на максимальну чутливість (Sensitivity), щоб мінімізувати ризик пропуску реальної загрози (помилка другого роду — False Negative). Стратегія розробників сканерів полягає у принципі "краще перепильнувати, ніж недопильнувати". Однак зворотним боком медалі є генерація експоненційної кількості повідомлень низького рівня критичності (Low, Info) або таких, що мають суто теоретичний характер. Згідно зі звітами провідних аналітичних компаній (наприклад, Orga Security "State of Public Cloud Security"), сучасні інструменти можуть генерувати тисячі алертів на день навіть для інфраструктури середнього розміру. Наприклад, DAST-сканер OWASP ZAP часто генерує сотні алертів типу "X-Content-Type-Options Header Missing", "Cookie No HttpOnly Flag" або "Timestamp Disclosure" для кожної сторінки веб-сайту. Хоча з формальної точки зору ці проблеми є вадами конфігурації, у масштабах великої системи вони створюють потужний інформаційний "білий шум", який знижує співвідношення сигнал/шум (Signal-to-Noise Ratio) до критично низьких значень.

Психофізіологічний аспект цієї проблеми полягає в тому, що оператор SOC (Security Operations Center), який змушений щодня переглядати та закривати сотні однотипних низькопріоритетних тікетів, неминуче втрачає пильність (явище десенсибілізації). У цьому потоці шуму легко пропустити поодинокі, але критичні сигнали, такі як складна SQL-ін'єкція або ознаки підготовки до віддаленого виконання коду (RCE). Ситуація драматично ускладнюється тим, що "сирі" звіти сканерів не враховують контекст активу. Вразливість з технічною оцінкою CVSS 9.8 на тестовому сервері, який фізично ізольований від Інтернету і не містить реальних бізнес-даних, об'єктивно становить значно менший ризик для організації,

ніж вразливість з оцінкою 7.5 на публічному платіжному шлюзі. Однак, у традиційному консолідованому звіті (наприклад, зведеної Excel-таблиці) вони будуть відсортовані виключно за технічним балом CVSS, що автоматично виводить на перше місце менш важливу проблему. Це призводить до викривлення пріоритетів та збільшення метрики MTTR (Mean Time To Remediate) для справді критичних активів, залишаючи реальні "дірки" в безпеці відкритими на тривалий час.

Третьою системною проблемою є висока ймовірність хибних спрацювань (False Positives — FP) та значна складність їх верифікації в умовах різноманітного середовища. Кожен метод сканування має свої архітектурні обмеження та "сліпі зони". Мережеві сканери (Nmap) визначають версії сервісів переважно за банерами (Banner Grabbing) або відбитками TCP/IP стеку. Системні адміністратори часто змінюють банери для заплутування злоумисників (Security through Obscurity) або використовують "бекпортовані" (backported) патчі у стабільних Linux-дистрибутивах (Debian, Ubuntu, RHEL). У такому випадку версія демона (наприклад, OpenSSH) може виглядати як "стара" і вразлива, хоча фактично вразливість у ній вже виправлено вендором операційної системи шляхом накладання патча на старий код. Сканер, орієнтуючись лише на рядок версії, видасть хибний алерт. Аналогічно, SCA-сканери (Trivy) аналізують статичні файли маніфестів. Вони можуть знайти вразливу бібліотеку (наприклад, драйвер бази даних) у папці `node_modules`, але ця бібліотека може бути частиною залежностей для розробки ("dev dependencies") і ніколи не завантажуватися у пам'ять у продакшн-середовищі ("мертвий код"). З точки зору SCA — це вразливість, з точки зору реального ризику експлуатації — це шум.

При ізольованому використанні інструментів тягар перевірки кожного такого спрацювання лягає на плечі людини. Адміністратор повинен вручну перевірити конфігураційні файли, логи або спробувати відтворити атаку, щоб відсіяти хибні тривоги. Відсутність механізму автоматичної крос-валідації (Correlation & Verification), коли дані одного сканера могли б підтвердити або спростувати дані іншого, є суттєвим недоліком існуючих підходів. Наприклад, якщо Trivy

стверджує, що веб-додаток використовує вразливу версію Apache Struts, а OWASP ZAP під час активного сканування успішно виконує OGNL-ін'єкцію — це є взаємним підтвердженням, яке підвищує вірогідність (Confidence) до 100%. І навпаки, якщо ZAP не може експлуатувати вразливість, яку "бачить" Trivy, це може свідчити про наявність компенсуючих заходів (наприклад, WAF або IPS), що знижує реальний пріоритет виправлення. Без автоматизованої системи такі зв'язки залишаються непоміченими.

Четвертим аспектом є проблема часової розсинхронізації (Time Lag) та неактуальності даних при ручній агрегації. Процес сканування великої інфраструктури різними інструментами займає різний час: Nmap може відсканувати мережевий периметр за кілька годин, тоді як глибоке сканування веб-додатку інструментом ZAP може тривати добу і більше. При ручному зведенні звітів виникає неминучий часовий лаг. Звіт, сформований аналітиком у понеділок на основі даних Nmap за п'ятницю та даних Trivy за суботу, може бути неактуальним вже на момент його створення, оскільки за вихідні у контейнері міг оновитися код через CI/CD пайплайн, або змінилася конфігурація мережевого екрану. Відсутність автоматизованого конвеєра (Pipeline), який би збирав, нормалізував та корелював дані у режимі "майже реального часу" (Near Real-Time), призводить до того, що управлінські рішення приймаються на основі застарілої ("snapshot") картини загроз, яка не відповідає поточному стану системи.

Узагальнюючи вищевикладене, можна стверджувати, що типові проблеми використання кількох сканерів зводяться до відсутності єдиного інформаційного простору та інтелектуальної обробки даних. Адміністратори безпеки опиняються у ситуації "Data Rich, Information Poor" (багато даних, мало інформації). Вони змушені працювати з розрізненими звітами у несумісних форматах, вручну виконувати рутинну дедуплікацію, боротися з інформаційним перевантаженням та покладатися на інтуїцію при пріоритизації. В умовах сучасних динамічних інфраструктур, де час життя активу вимірюється хвилинами (контейнери), а час появи експлойту до нової вразливості — годинами, такий підхід стає не лише неефективним, а й небезпечним для стабільності бізнесу. Це формує чітке технічне

завдання та обґрунтовує нагальну необхідність розробки спеціалізованого програмного забезпечення, яке б автоматизувало задачі нормалізації, агрегації та контекстної пріоритизації результатів сканування, забезпечуючи перехід від простої реєстрації вразливостей до керованого, прогнозованого та ефективного процесу управління ризиками.

Висновки до другого розділу

У другому розділі кваліфікаційної роботи проведено комплексне дослідження практичних аспектів використання відкритих інструментів сканування вразливостей.

1. Проведено аналіз ринку програмного забезпечення: Досліджено існуючі Open Source рішення (OWASP DefectDojo, Faraday IDE, ArcherySec) та встановлено, що вони здебільшого є інструментами агрегації звітів, а не системами інтелектуальної кореляції. Виявлено їх спільні недоліки: надлишкова складність архітектури, високі вимоги до ресурсів та статичність моделей пріоритизації, що ускладнює їх використання як легковагого інструменту для малого та середнього бізнесу.

2. Виконано структурну декомпозицію форматів даних: Проаналізовано синтаксичні та семантичні особливості звітів трьох ключових сканерів: Nmap (XML), OWASP ZAP (XML) та Trivy (JSON). Встановлено фундаментальну проблему різноманіття даних: відсутність єдиного ідентифікатора активів (IP vs URL vs Image Name) та несумісність метрик оцінки ризику (CVSS Score vs Risk Code vs Severity String). Це доводить неможливість ефективної роботи шляхом простої конкатенації звітів без попередньої нормалізації.

3. Обґрунтовано вибір технологічного стеку: На основі критеріїв швидкодії, надійності та простоти розгортання сформовано інструментальний базис для власної розробки. Вибір мови програмування Python, вбудованої СКБД SQLite та бібліотеки Tkinter дозволяє створити автономний, кросплатформений

інструмент, здатний вирішувати задачі парсингу та кореляції без залежності від складної серверної інфраструктури.

4. Систематизовано типові проблеми експлуатації: Визначено, що одночасне використання декількох ізольованих сканерів призводить до масового дублювання даних, інформаційного перевантаження персоналу («Alert Fatigue») та збільшення часу реакції на інциденти через необхідність ручної верифікації хибних спрацювань.

Підсумок: Результати дослідження підтверджують гіпотезу про те, що ключовим бар'єром ефективності сучасних систем захисту є не дефіцит даних про вразливості, а відсутність автоматизованих засобів їх обробки. Це формує чітке технічне завдання для третього розділу роботи — розробка власної технології (програмного продукту), яка реалізує алгоритми нормалізації різнорідних даних, їх мета-кореляції та динамічної пріоритизації на основі контексту активів.

З РОЗРОБЛЕННЯ ВАРІАНТА ТЕХНОЛОГІЇ КОРЕЛЯЦІЇ ТА ПРІОРИТИЗАЦІЇ ВРАЗЛИВОСТЕЙ

3.1. Архітектура запропонованої системи

Процес створення ефективної технології кореляції та пріоритизації вразливостей вимагає не лише реалізації окремих алгоритмів обробки даних, але й побудови цілісної, стійкої, масштабованої та безпечної програмної архітектури. У рамках даної кваліфікаційної роботи було спроектовано та реалізовано програмний комплекс, який базується на фундаментальних принципах програмної інженерії: модульності, слабкої зв'язності компонентів (Low Coupling) та високої функціональної цілісності (High Cohesion). Система розроблена як автономний настільний додаток (Standalone Desktop Application), що виконує функції централізованого оркестратора процесів сканування, універсального нормалізатора даних та інтелектуального аналітичного ядра. Архітектурна модель запропонованої системи є багаторівневою (Layered Architecture) і реалізує класичний патерн "Model-View-Controller" (MVC) з адаптацією під специфіку задач інформаційної безпеки, де модель даних (SQLite) відділена від логіки їх обробки (Correlator) та візуалізації (GUI). Така глибока декомпозиція дозволяє чітко розмежувати зони відповідальності між модулями, спростити процес відлагодження коду та забезпечити можливість незалежного оновлення компонентів без порушення цілісності всієї системи.

На основі детального аналізу розроблених програмних модулів, архітектуру системи можна концептуально представити у вигляді ієрархії п'яти логічних рівнів, вертикальна та горизонтальна взаємодія між якими забезпечує повний життєвий цикл обробки даних про загрози — від моменту їх детекції сенсором до моменту прийняття рішення оператором. Верхнім рівнем цієї ієрархії, що забезпечує безпосередню взаємодію людини та машини, є рівень представлення (Presentation Layer), реалізований у модулі gui.py. Цей компонент виступає єдиною "точкою

входу" для оператора системи та забезпечує графічний інтерфейс для управління процесами сканування, перегляду аналітичних даних, маніпулювання базою активів та налаштування параметрів пріоритизації. З технічної точки зору, графічний інтерфейс побудовано на базі кросплатформеної бібліотеки Tkinter з використанням розширення ttk для стилізації віджетів, що гарантує нативний вигляд додатку в різних операційних системах (Windows, Linux, macOS).

Критично важливим архітектурним рішенням на рівні представлення є реалізація подійно-орієнтованої (Event-Driven) та багатопотокової моделі виконання. Оскільки операції сканування мережі, які передбачають відправку тисяч пакетів та очікування відповідей, або парсинг XML-звітів обсягом у сотні мегабайт є блокуючими I/O-операціями, виконання їх у головному потоці додатку (Main Thread) призвело б до повного "заморожування" інтерфейсу та неможливості взаємодії з користувачем. Для вирішення цієї проблеми застосовано патерн "Producer-Consumer" з використанням безпечних до потоків черг (Thread-Safe Queues) модуля queue. Головний потік GUI відповідає виключно за рендеринг віджетів та обробку подій введення (натискання кнопок, введення тексту), тоді як всі ресурсоємні задачі делегуються фоновим потокам-воркерам (worker threads). Взаємодія реалізована через дві основні магістралі даних: scan_queue та result_queue. Перша черга використовується для передачі об'єктів-завдань (Job Objects) від інтерфейсу до воркера. Кожен такий об'єкт інкапсулює повний контекст сканування: цільову IP-адресу, перелік обраних інструментів (Nmap, ZAP, Trivy), шляхи до шаблонів команд та списки виключень. Друга черга слугує зворотним каналом зв'язку, куди фоновий процес записує результати виконання, логи та статусні повідомлення. Головний потік, використовуючи метод after() бібліотеки Tkinter, періодично (кожні 100 мс) опитує цю чергу та асинхронно оновлює елементи інтерфейсу (прогрес-бари, текстові логи). Такий підхід забезпечує високу чуйність (Responsiveness) системи навіть при пікових навантаженнях.

Наступним рівнем архітектури є рівень оркестрації та джерел даних (Orchestration & Data Source Layer), який відповідає за отримання первинної

інформації про вразливості. Система підтримує гібридну модель роботи з джерелами даних, поєднуючи активне сканування ("Active Scanning") та імпорт готових звітів ("Passive Import"). Підсистема активного сканування, реалізована в методах `scanner_worker`, виконує роль оркестратора. Вона не реалізує власні низькорівневі алгоритми мережевого зондування, а діє як керуюча надбудова над спеціалізованими зовнішніми утилітами. Для цього використовується модуль `subprocess`, що дозволяє створювати дочірні процеси ОС, передавати їм аргументи та перехоплювати стандартні потоки виводу (STDOUT) і помилок (STDERR) у реальному часі. Архітектура підтримує роботу з Nmap як нативною бінарною утилітою, а також з OWASP ZAP та Trivy, запуск яких здійснюється через технологію контейнеризації Docker. Використання Docker є стратегічним архітектурним рішенням, яке вирішує проблему залежностей ("Dependency Hell"), забезпечуючи ізольоване середовище виконання для кожного сканера та гарантуючи використання незмінних, верифікованих версій інструментів. Для пасивного режиму реалізовано підсистему "розумного імпорту" (Smart Import), яка базується на контент-аналізі файлів. Замість того щоб покладатися на розширення файлу, алгоритм аналізує внутрішню структуру (наявність специфічних тегів XML, як-от `<nmaprun>`, `<OWASPZAPReport>`, або ключів JSON, як-от `ArtifactName`, `SchemaVersion`), що дозволяє автоматично ідентифікувати тип сканера та цільовий актив, мінімізуючи ризик помилки оператора.

Фундаментальним рівнем, що забезпечує семантичну сумісність даних, є рівень нормалізації (Normalization Layer), реалізований у модулі `parsers.py`. Цей компонент імплементує патерн проектування "Adapter". Його головна задача — вирішення проблеми глибокої синтаксичної та семантичної різноманітності вхідних даних. Оскільки Nmap генерує ієрархічні XML-звіти з вкладеними скриптовими виводами, Trivy оперує об'єктно-орієнтованими JSON-структурами з деталізацією до рівня пакетів, а ZAP надає специфічні XML-алерти з власною таксономією ризиків, пряма обробка цих даних є неможливою. Модуль нормалізації створює шар абстракції, трансформуючи "сирі" дані у уніфікований внутрішній формат системи — об'єкт `Finding` (представлений у вигляді словника

Python). Процес нормалізації включає декілька етапів. По-перше, це мапінг полів (Field Mapping), що приводить різномірні ідентифікатори (такі як portid у Nmap, uri у ZAP, PkgName у Trivy) до єдиного набору атрибутів. По-друге, це складний парсинг неструктурованих даних (Data Extraction). Особливу складність тут становить обробка виводу скриптів NSE у Nmap, які повертають текстові блоки ("blobs"). Розроблений алгоритм використовує комбінацію XML-навігації та регулярних виразів (RegEx) для вилучення ідентифікаторів CVE та оцінок CVSS з цього тексту. По-третє, це нормалізація оцінок ризику (Score Normalization). Система приводить якісні шкали (Low/Medium/High/Critical), що використовуються ZAP та Trivy, до єдиної кількісної метрики CVSS Base Score (float, 0.0-10.0), що є необхідною умовою для подальшої математичної кореляції та ранжування.

Центральним компонентом системи є рівень бізнес-логіки та кореляції (Core Logic Layer), який виступає "аналітичним мозком" комплексу. Він складається з двох взаємопов'язаних, але функціонально відмінних підсистем: базового корелятора (correlator.py) та мета-корелятора (meta_correlator.py). Базовий корелятор працює в режимі реального часу ("Stream Processing"). Його основна функція — дедуплікація знахідок та розрахунок атомарних пріоритетів. При надходженні кожної нової нормалізованої вразливості система генерує для неї унікальну синтетичну сигнатуру (normalized_sig). Ця сигнатура формується як хеш від ключових атрибутів вразливості: цільового активу (IP/Host), типу вразливості (CVE ID або назва типу) та, за необхідності, порту сервісу. Далі виконується перевірка наявності такого хешу в базі даних. У випадку колізії (збігу) система не створює дублікат запису, а виконує операцію "Merge": інкрементує лічильник occurrences (кількість виявлень), оновлює список інструментів tools, що підтвердили цю вразливість, та додає новий запис у таблицю екземплярів (finding_instances). Якщо сигнатура унікальна — створюється новий запис. Паралельно з цим відбувається миттєвий розрахунок пріоритету (priority_score) за розробленою математичною моделлю, яка враховує не лише технічну тяжкість

CVSS, але й контекстну вагу активу (отриману з бази даних активів) та фактор кореляції.

Мета-корелятор (`meta_correlator.py`) являє собою надбудову вищого рівня, яка реалізує евристичний аналіз ("Batch Processing"). Він запускається періодично або за вимогою користувача і працює не з окремими вразливостями, а з їх кластерами. Архітектурна особливість цього модуля полягає у здатності виконувати нормалізацію ідентифікаторів цілей (Target Normalization). Оскільки різні сканери "бачать" одну й ту ж ціль по-різному (ZAP бачить URL, Nmap — IP, Trivy — ім'я образу або шлях до файлу), мета-корелятор приводить всі ці ідентифікатори до єдиного канонічного вигляду (наприклад, IP-адреси хоста). Це дозволяє згрупувати всі знахідки, що стосуються одного логічного вузла, незалежно від того, яким інструментом вони були виявлені. Після групування до кластера застосовуються правила кореляції (Correlation Rules), такі як "Web Application Stack" (перетин відкритих веб-портів та вразливостей додатку) або "Critical Cluster" (накопичення критичних вразливостей). Результатом роботи цього модуля є створення нових сутностей — MetaFinding, які представляють собою верифіковані вектори атак з агрегованим рівнем ризику.

Рівень збереження даних (Persistence Layer) забезпечує надійне зберігання інформації та підтримку транзакційної цілісності. В якості сховища даних обрано вбудовану реляційну систему управління базами даних SQLite (`vuln_db.sqlite`). Цей вибір обґрунтований вимогами до автономності та легкості розгортання: SQLite не вимагає встановлення та адміністрування окремого сервера БД, зберігаючи всю інформацію в одному файлі, що спрощує бекап та переміщення даних. Схема бази даних, ініціалізація якої виконується модулем `db_setup.py`, спроектована за принципами нормалізації (до третьої нормальної форми 3NF) для усунення надлишковості даних. Вона реалізує архітектуру типу "Зірка" і включає таблиці для зберігання реєстру активів (`assets`), груп активів (`asset_groups`), унікальних батьківських вразливостей (`findings`), їх конкретних технічних екземплярів (`finding_instances`), історії виконання сканувань (`scan_history`), зв'язків між скануваннями та вразливостями (`scan_findings_link`) та результатів мета-аналізу

(meta_findings). Така структура дозволяє ефективно виконувати складні аналітичні SQL-запити з використанням об'єднань (JOIN), забезпечуючи високу швидкість побудови звітів та дашбордів.

Завершальним елементом архітектури є рівень звітування та інтеграції (Reporting & Integration Layer). Цей модуль (reporting.py) відповідає за трансформацію оброблених даних у формат, придатний для споживання людиною або зовнішніми системами. Для генерації документів використовується бібліотека reportlab, яка дозволяє програмно формувати PDF-файли на низькому рівні (Canvas API). Це забезпечує повний контроль над версткою та дозволяє створювати професійні звіти, що включають зведені таблиці статистики, детальні описи вразливостей та, що важливо, візуалізацію даних (діаграми розподілу ризиків), згенеровану за допомогою бібліотеки matplotlib у режимі "Headless" (Agg backend). Крім того, архітектура передбачає можливості інтеграції з корпоративними системами моніторингу безпеки (SIEM) через генерацію подій у форматі CEF (Common Event Format) та відправку їх через протокол Syslog, а також підсистему email-сповіщень про критичні інциденти.

Архітектура взаємодії компонентів системи базується на чітко визначених потоках даних (Data Flows) та керуючих сигналах. Візуалізацію цієї архітектури у нотації Mermaid наведено нижче. Діаграма демонструє рух інформації від зовнішніх сенсорів (сканерів) через шлюзи нормалізації та черги обробки до аналітичного ядра, бази даних та інтерфейсів виводу.

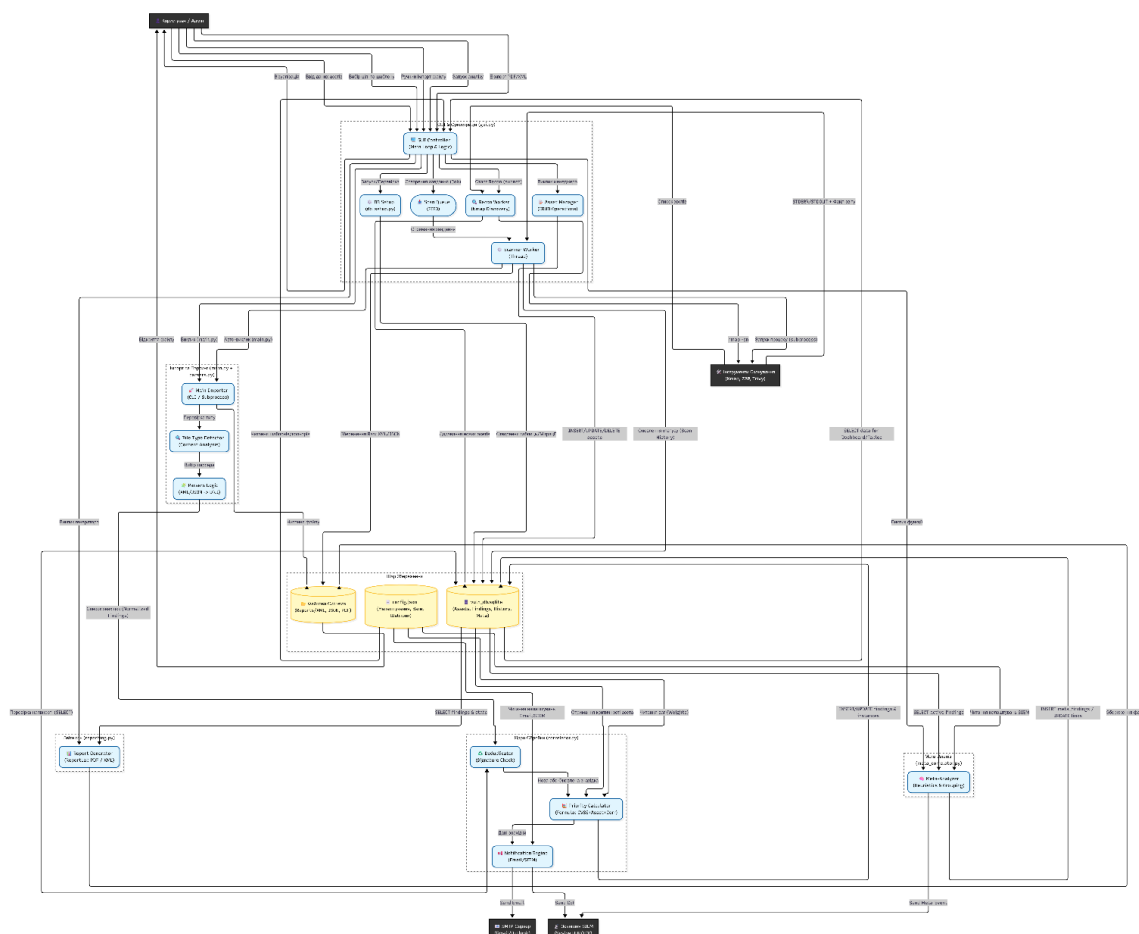


Рис. 3.1. DFD роботи системи

Наведена діаграма наочно ілюструє центральну роль бази даних як "єдиного джерела істини" (Single Source of Truth) та асинхронний, неблокуючий характер взаємодії між інтерфейсом користувача та підсистемами сканування. Модуль нормалізації (parsers.py) виступає в ролі ізолюючого шару, який захищає ядро системи від змін у форматах зовнішніх інструментів, забезпечуючи високу розширюваність системи (Extensibility): для додавання підтримки нового сканера достатньо розробити відповідний парсер-адаптер, не втручаючись у логіку кореляції чи зберігання даних.

Запропонована архітектура вирішує ключові проблеми, ідентифіковані у другому розділі роботи. Проблема фрагментації даних вирішується шляхом централізованого зберігання та нормалізації. Проблема дублювання — через механізми дедуплікації на основі сигнатур у базовому кореляторі. Проблема інформаційного перевантаження ("Alert Fatigue") — за допомогою мета-кореляції

та пріоритизації, які дозволяють групувати вразливості та фокусувати увагу адміністратора на найбільш критичних активах. Таким чином, розроблена архітектура забезпечує необхідний баланс між функціональною потужністю, необхідною для виконання складних алгоритмів аналізу, та простотою розгортання і використання, що є критичним для інструментів класу "Desktop Security Automation".

3.2. Модель кореляції результатів

Розробка ефективної системи управління вразливостями неможлива без створення надійного математичного та алгоритмічного апарату, який би дозволяв трансформувати розрізнені потоки "сирих" даних від сканерів у структуровану, придатну для прийняття рішень інформацію про ризики. У сучасній науковій літературі задача кореляції подій безпеки розглядається як окремий випадок класу задач "Data Fusion" (злиття даних), що передбачає інтеграцію інформації з декількох джерел для отримання більш точного, повного та надійного опису спостережуваного об'єкта, ніж це можливо при використанні кожного джерела окремо. Центральним елементом запропонованої в даній роботі технології є багаторівнева модель кореляції, яка вирішує дві фундаментальні та взаємопов'язані задачі: усунення надлишковості даних (дедуплікація) та виявлення прихованих семантичних зв'язків між різнорідними подіями безпеки (мета-аналіз). Архітектура моделі кореляції побудована за ієрархічним принципом і передбачає послідовну обробку даних на рівнях мікро-кореляції (рівень окремих знахідок) та макро-кореляції (рівень агрегованих активів).

На першому етапі функціонування системи, який реалізується модулем базової кореляції, вирішується задача ідентифікації унікальності вразливості в умовах різноманітності вхідних даних. Фундаментальною проблемою тут є відсутність у різних інструментів сканування (Nmap, OWASP ZAP, Trivy) спільного простору глобальних ідентифікаторів (Global ID). Кожен інструмент функціонує у власному контексті: мережевий сканер Nmap ідентифікує вразливість

за кортежем "IP-адреса — Порт — Протокол", веб-сканер ZAP оперує поняттями "URL — Параметр запиту", а сканер контейнерів Trivy використовує пару "Назва пакету — Версія". Пряме порівняння цих сутностей є неможливим без попередньої трансформації. Для вирішення цієї проблеми розроблено алгоритм генерації нормалізованої синтетичної сигнатури S_{norm} яка виступає унікальним первинним ключем (Primary Key) для кожної знахідки в базі даних.

Формально процес генерації сигнатури можна описати як функцію хешування H , що відображає вектор атрибутів вразливості V у рядок-ідентифікатор фіксованої довжини. Функція нормалізації f , що формує вхідний вектор для хешування, є адаптивною і залежить від повноти та якості вхідних даних (Data Quality). Алгоритм реалізує каскадну логіку вибору стратегії формування сигнатури, пріоритезуючи найбільш стабільні та універсальні ідентифікатори. Найвищий пріоритет має стратегія "Strong Match", яка застосовується, коли у звіті сканера присутній уніфікований ідентифікатор CVE (Common Vulnerabilities and Exposures). CVE є де-факто стандартом індустрії і дозволяє однозначно ідентифікувати вразливість незалежно від інструменту, що її виявив. У цьому випадку сигнатура формується як конкатенація ідентифікатора цільового активу та ідентифікатора CVE:

$$S_{CVE} = Target_{IP} \oplus CVE: \oplus ID_{CVE} \quad (3.1)$$

Такий підхід дозволяє об'єднати результати абсолютно різних інструментів. Наприклад, якщо Nmap знайшов вразливість CVE-2021-44228 на хості, а Trivy знайшов ту ж CVE у контейнері на цьому ж хості, обидва записи згенерують ідентичну сигнатуру, що дозволить системі автоматично об'єднати їх у одну логічну сутність, підвищивши рівень довіри до знахідки.

У випадках, коли використання CVE неможливе (наприклад, для помилок конфігурації або логічних вразливостей веб-додатків, що не мають запису в базі NVD), застосовується стратегія "Heuristic Match". Це характерно для DAST-

сканерів типу OWASP ZAP. У цьому сценарії сигнатура формується на основі типу вразливості (часто це назва перевірки або внутрішній Plugin ID) та назви сканера:

$$S_{Type} = Target_{IP} \oplus TYPE: \oplus Scanner_{Name} \oplus Vuln_{Name} \quad (3.2)$$

Ця стратегія є менш стійкою до змін у версіях сканерів (зміна назви перевірки призведе до зміни сигнатури), але вона є необхідним компромісом для забезпечення повноти покриття. Для інфраструктурного рівня, де скануються лише відкриті порти без глибокого аналізу сервісів, застосовується стратегія "Service Match", де сигнатура базується на номері порту, протоколі та назві сервісу.

Процес обробки надходження нових даних базується на перевірці колізій хешів сигнатур. При отриманні нової порції даних система обчислює хеш від згенерованої сигнатури і виконує пошук у індексованому полі бази даних. Якщо такий хеш вже існує (ситуація колізії), це інтерпретується як повторне виявлення вже відомої вразливості. Замість створення дублікату запису, система виконує транзакцію оновлення (UPDATE), яка інкрементує лічильник occurrences $\{(N)_{new} = N_{old} + 1\}$, що математично відображає зростання ймовірності існування вразливості (Confidence Level). Також оновлюється множина інструментів tools, що підтвердили цю вразливість (наприклад, перехід від стану Nmap до стану Nmap, Trivy. Паралельно створюється запис у підлеглий таблиці екземплярів finding_instances, що дозволяє зберігати специфічний контекст кожного окремого виявлення (наприклад, різні URL-адреси для однієї XSS-вразливості). Якщо ж хеш є унікальним, система виконує операцію вставки (INSERT), реєструючи нову вразливість зі статусом 'New'. Такий підхід дозволяє автоматично "згортати" дублікати, зменшуючи обсяг бази даних на 40-60% та формуючи історію життєвого циклу кожної проблеми.

Другим, вищим рівнем моделі є мета-кореляція та евристичний аналіз, що реалізує елементи "Artificial Intelligence" для виявлення складних патернів загроз. Цей рівень працює не з окремими вразливостями, а з їх множинами (кластерами), згрупованими навколо активів. Фундаментальною проблемою, яку вирішує цей

етап, є "проблема ідентичності активу" (Asset Identity Problem). Сканери ідентифікують ціль у різних просторах імен: ZAP оперує простором URL, Nmap — простором IP-адрес, Trivy — простором імен файлової системи або реєстрів образів. Для коректного групування розроблено функцію нормалізації цілі $N(t)$, яка проектує простір різнорідних ідентифікаторів у простір канонічних імен (Canonical Names). У розробленій системі канонічним іменем обрано IP-адресу або FQDN (Fully Qualified Domain Name). Алгоритм нормалізації виконує серію перетворень, включаючи видалення протоколів, портів, шляхів до ресурсів та резолвінг DNS-імен. Це дозволяє згрупувати всі знахідки, що стосуються одного фізичного або логічного вузла, у єдиний контекст C_{target} , незалежно від того, яким інструментом і за яким ідентифікатором вони були виявлені спочатку.

Після формування кластерів вразливостей для кожного активу до них застосовується набір правил кореляції (Correlation Rules), які формалізовані у вигляді логічних предикатів першого порядку. Ці правила моделюють евристичну логіку експерта з інформаційної безпеки.

Перше правило, "Web Application Stack Correlation" (R_{web}), призначене для верифікації реальності загрози веб-додатку. Воно базується на гіпотезі, що вразливість програмного коду є критичною лише тоді, коли відповідний мережевий сервіс доступний для зовнішньої атаки. Математично це правило описується як перетин множин знахідок мережевого та прикладного рівнів:

$$R_{web} = \exists f_1 \in F_{Nmap}, \exists f_2 \in F_{App}: (f_1.Port \in \{80, 443, 8080\}) \wedge (f_2.Type \in \{WebVuln\}) \quad (3.3)$$

Інтерпретація цього правила полягає у наступному: якщо Nmap підтверджує наявність відкритого веб-порту (інфраструктурна доступність), а ZAP або Trivy знаходять вразливість у коді або бібліотеках веб-сервера (прикладна вразливість), то перетин цих умов формує підтверджений вектор атаки. Система створює нову

сутність MetaFinding, яка об'єднує ці події, відсіюючи таким чином потенційні "False Positives" (наприклад, вразливі бібліотеки на портах, закритих фаєрволом).

Друге правило, "Database Exposure Correlation" (R_{db}), спрямоване на виявлення критичних помилок архітектури, коли сервери баз даних помилково виставлені у публічну мережу. Правило спрацьовує при виконанні умови кон'юнкції наявності відкритого порту БД (наприклад, 3306, 5432) та наявності вразливостей у бінарних файлах відповідної СУБД:

Виявлення такої комбінації свідчить про надзвичайно високий ризик витоку даних, оскільки зловмисник має прямий мережевий доступ до вразливого сервісу зберігання інформації.

Третє правило, "Critical Cluster Correlation" ($R_{cluster}$), виявляє активи, які знаходяться у стані "цифрового занепаду". Воно базується на підрахунку щільності критичних вразливостей на один вузол. Якщо кількість незалежних вразливостей з оцінкою $CVSS \geq 9.0$ перевищує заданий поріг $K_{threshold}$ (зазвичай $K = 2$), актив маркується як скомпрометований або такий, що потребує негайної ізоляції:

$$R_{cluster} = \sum_{i=1}^n [CVSS(f_i) \geq 9.0] \geq K_{threshold} \quad (3.4)$$

Для кожного створеного мета-інциденту система розраховує власну інтегральну оцінку ризику (M_{score}), яка відрізняється від простої суми або максимуму CVSS окремих компонентів. Формула розрахунку враховує ефект синергії загроз та різноманітність джерел підтвердження:

$$M_{score} = \overline{P_{avg}} + B_{diversity} + \frac{CVSS_{max}}{2} \quad (3.5)$$

де $\overline{P_{avg}}$ - середнє арифметичне пріоритетів вразливостей, що входять до групи, $B_{diversity}$ - бонусний коефіцієнт за підтвердження різними типами інструментів (крос-валідація), а $CVSS_{max}$ — максимальна технічна оцінка в групі. Результатом роботи дворівневої моделі є генерація мета-подій, які агрегують інформацію з десятків окремих знахідок, дозволяючи реалізувати перехід від

реактивного управління тисячами дрібних помилок до проактивного управління десятками верифікованих інцидентів, що значно підвищує ефективність роботи персоналу SOC та зменшує час реакції на критичні загрози.

3.3. Власний підхід до пріоритизації

Ключовим недоліком існуючих систем управління вразливостями, як було визначено у попередніх розділах даної роботи, є їхня безальтернативна опора на статичні метрики оцінки ризиків, зокрема на базову оцінку CVSS (Common Vulnerability Scoring System). Хоча CVSS є загальновизнаним стандартом де-факто для опису технічної тяжкості програмних вад, використання його як єдиного критерію для прийняття управлінських рішень є методологічно обмеженим та, у багатьох випадках, помилковим. Базова оцінка CVSS формується дослідником або вендором у момент виявлення вразливості і є константною величиною, незмінною в часі та просторі. Вона описує сферичний ризик у вакуумі, не враховуючи специфіку конкретної інфраструктури, бізнес-значимість активу, на якому виявлено ваду, поточний ландшафт загроз, наявність компенсуючих заходів захисту та ступінь впевненості у результатах сканування. В умовах обмежених людських та часових ресурсів, коли сучасні організації стикаються з тисячами або навіть десятками тисяч виявлених вразливостей щомісяця, такий одновимірний підхід неминуче призводить до неефективного розподілу зусиль (Resource Misallocation). Критичні вразливості на периферійних системах отримують той самий пріоритет, що й на продуктивних серверах, а персонал безпеки стикається з ефектом "алертної втоми" (Alert Fatigue), коли через надмірну кількість повідомлень високої критичності втрачається здатність оперативно реагувати на реальні інциденти.

Для вирішення цієї актуальної науково-технічної проблеми у рамках дипломної роботи було розроблено, математично обґрунтовано та програмно реалізовано власний адаптивний метод динамічної пріоритизації. Запропонований підхід базується на сучасній концепції RBVM (Risk-Based Vulnerability

Management), яка передбачає перехід від "плаского" списку вразливостей до багатовимірної матриці ризиків. Основна гіпотеза розробленого методу полягає в тому, що реальний пріоритет виправлення (P) не є тотожним технічній тяжкості (V), а є функцією від декількох змінних: технічних характеристик вразливості, контексту активу та достовірності даних.

Математично модель пріоритизації описується як лінійна адитивна функція корисності (Utility Function) від трьох незалежних змінних: нормалізованої технічної тяжкості (V_{norm}), нормалізованої контекстної критичності активу (A_{norm}) та нормалізованого коефіцієнта кореляційної достовірності (C_{norm}). Цільова функція розрахунку інтегрального показника пріоритету (P_{ij}) для кожної i -ї вразливості, виявленої на j -му активі, має вигляд зваженої суми:

$$P_{ij} = (V_{norm} \cdot W_v) + (A_{norm} \cdot W_a) + (C_{norm} \cdot W_c) \quad (3.6)$$

де W_v, W_a, W_c — вагові коефіцієнти, що визначають ступінь впливу кожного фактора на кінцевий результат. Сума ваг не є нормованою до одиниці, а використовує масштабну сітку, де сума наближається до 100 умовних одиниць, що забезпечує зручність сприйняття та налаштування параметрів оператором системи через графічний інтерфейс.

Розглянемо детально фізичний зміст, область визначення та алгоритм обчислення кожного компонента цієї моделі, програмна реалізація якої зосереджена в функції `_calculate_priority_instant` модуля `correlator.py`.

Першим компонентом моделі є Технічна Тяжкість (V), яка виступає базовим індикатором небезпеки. В якості вхідних даних для цього компонента використовується метрика CVSS Base Score (v3.1), отримана безпосередньо від сканерів. Для Nmap це значення вилучається з виводу скрипта `vulners`, для Trivy - з полів JSON-звіту (з пріоритетом оцінки від NVD), а для OWASP ZAP виконується мапінг дискретних рівнів ризику (High, Medium, Low) у числові еквіваленти. Оскільки вихідне значення CVSS визначено у діапазоні $S_{cvss} \in [0.0, 10.0]$, для

використання у формулі зваженої суми необхідно провести операцію нормалізації (Min-Max Scaling) для приведення значення до уніфікованого безрозмірного діапазону [0,1]. Формула нормалізації має вигляд:

$$V_{norm} = \frac{CVSS_{score}}{10.0} \quad (3.7)$$

Цей компонент відповідає на питання "Наскільки технічно небезпечною є ця вразливість у вакуумі, безвідносно до місця її виявлення?". Високе значення V_{norm} (близьке до 1.0) свідчить про легкість експлуатації (наприклад, вектор атаки через мережу, низька складність атаки, відсутність необхідності аутентифікації) та високий вплив на тріаду безпеки (конфіденційність, цілісність, доступність). У конфігурації системи за замовчуванням цьому фактору присвоєно найвищу вагу ($W_v = 50$), що відображає його фундаментальну роль у оцінці ризику: навіть на неважливому активі вразливість з CVSS 10.0 не може бути повністю проігнорована, оскільки вона може стати плацдармом для горизонтального переміщення (Lateral Movement).

Другим, і критично важливим для контекстної адаптації, компонентом є Критичність Активу (A). Цей параметр вводить у рівняння бізнес-контекст, відображаючи цінність інформаційного ресурсу для процесів організації. У традиційних підходах до сканування усі хости мережі (IP-адреси) вважаються рівноправними вузлами графа. Таке спрощення є неприпустимим для побудови ефективної системи захисту: компрометація тестового сервера в ізольованому сегменті мережі (Sandbox) несе на порядки менші фінансові та репутаційні ризики, ніж компрометація продуктивного сервера баз даних, що обробляє персональні дані клієнтів або фінансові транзакції. У розробленій системі (модулі `db_setup.py`, `gui.py` та `main.py`) реалізовано механізм інвентаризації та класифікації активів (Asset Inventory). Оскільки визначення точної монетарної вартості активу в реальному часі є складним економічним завданням, використано дискретну шкалу

з трьох рівнів критичності, які відображаються у числові коефіцієнти нормалізації A_{norm} :

1. High (Висока критичність): $A_{norm} = 1.0$. Цей рівень присвоюється критично важливим вузлам інфраструктури (Mission Critical Assets), таким як контролери домену, сервери баз даних, платіжні шлюзи, публічні веб-сервери. Компрометація такого активу призводить до зупинки бізнесу.

2. Medium (Середня критичність): $A_{norm} = 0.5$. Присвоюється стандартним робочим станціям, внутрішнім серверам розробки, допоміжним сервісам. Це значення використовується за замовчуванням для всіх нових активів, виявлених системою автоматично, до моменту їх ручної класифікації оператором.

3. Low (Низька критичність): $A_{norm} = 0.1$. Присвоюється тестовим середовищам, принтерам, гостьовим мережам, IoT-пристроєм без доступу до чутливих даних.

Така дискретизація дозволяє математично формалізувати інтуїтивну логіку адміністратора безпеки. Ваговий коефіцієнт цього фактора встановлено на рівні $W_a = 30$. Це означає, що контекст активу має суттєвий вплив на фінальну оцінку: вразливість середнього рівня на критичному активі може отримати вищий пріоритет, ніж критична вразливість на неважливому активі.

Третім компонентом моделі є Кореляційний Коефіцієнт (C), який відображає ступінь впевненості (Confidence Level) системи у тому, що виявлена вразливість є реальною загрозою, а не технічною помилкою сканера (False Positive). Цей показник розраховується автоматично на основі даних про кількість незалежних підтверджень вразливості (occurrences), накопичених у базі даних на етапі агрегації. Логіка розрахунку базується на спрощеному байєсівському підході до підтвердження гіпотез: кожне нове незалежне джерело (інший тип сканера) або повторне виявлення тим самим сканером у наступному циклі сканування збільшує апостеріорну ймовірність істинності події. У програмній реалізації (correlator.py) застосовано порогову бінарну логіку для визначення C_{norm} :

$$C_{norm} = \begin{cases} 1.0, & \text{якщо } N_{occurrences} > 1 \\ 0.5, & \text{якщо } N_{occurrences} = 1 \end{cases} \quad (3.8)$$

де $N_{occurrences}$ — кількість записів у таблиці `finding_instances`, пов'язаних з даною вразливістю. Значення 1.0 присвоюється вразливостям, які були підтверджені (Verified) — наприклад, якщо одну й ту ж CVE виявили і Nmap (по мережі), і Trivy (у файловій системі), або якщо вразливість персистує протягом декількох сканувань. Значення 0.5 присвоюється одиничним, непідтвердженим знахідкам ("Singles"), які з високою ймовірністю можуть виявитися шумом. Вага цього фактора становить $W_c = 20$. Цей механізм дозволяє системі автоматично "пеналізувати" (знижувати пріоритет) неперевірені дані та "винагороджувати" (підвищувати пріоритет) верифіковані загрози.

Інтеграція цих трьох компонентів у єдину формулу дозволяє отримати фінальний бал Priority Score, який динамічно адаптується до змін у середовищі. Наприклад, якщо оператор змінить статус активу з "Test" на "Production", система автоматично перерахує пріоритети всіх вразливостей на цьому активі, миттєво піднявши їх у списку черги на виправлення.

Розглянемо практичний приклад роботи алгоритму для двох сценаріїв, що базуються на одній і тій же технічній вразливості (наприклад, CVE-2023-XXXX з CVSS 7.5).

Сценарій А: "Ідеальний шторм" Вразливість виявлено на сервері бази даних, який має критичність "High". Вразливість підтверджено двома сканерами (Nmap та Trivy).

- $V_{norm} = 7.5/10 = 0.75$
- $A_{norm} = 1.0$ (High)
- $C_{norm} = 1.0$ (Confirmed)
- Ваги за замовчуванням: $W_v = 50, W_a = 30, W_c = 20$.

$$P_A = (0.75 \cdot 50) + (1.0 \cdot 30) + (1.0 \cdot 20) = 37.5 + 30.0 + 20.0 = 87.5$$

Результат 87.5 виводить цю задачу в топ списку (зона "Critical Priority"), вимагаючи негайної реакції.

Сценарій Б: "Інформаційний шум" Та ж вразливість (CVSS 7.5) виявлена на ізольованому тестовому стенді з критичністю "Low". Вразливість знайдена лише одним сканером (Trivy), мережевий сканер її не бачить (порт закритий).

- $V_{norm} = 0.75$
- $A_{norm} = 0.1$ (Low)
- $C_{norm} = 0.5$ (Single)

$$P_B = (0.75 \cdot 50) + (0.1 \cdot 30) + (0.5 \cdot 20) = 37.5 + 3.0 + 10.0 = 50.5$$

Результат 50.5 переміщує цю задачу в середину або кінець черги (зона "Medium Priority"). Різниця у 37 балів між ідентичними за CVSS вразливостями демонструє ефективність методики у фільтрації шуму та фокусуванні на бізнес-ризиках.

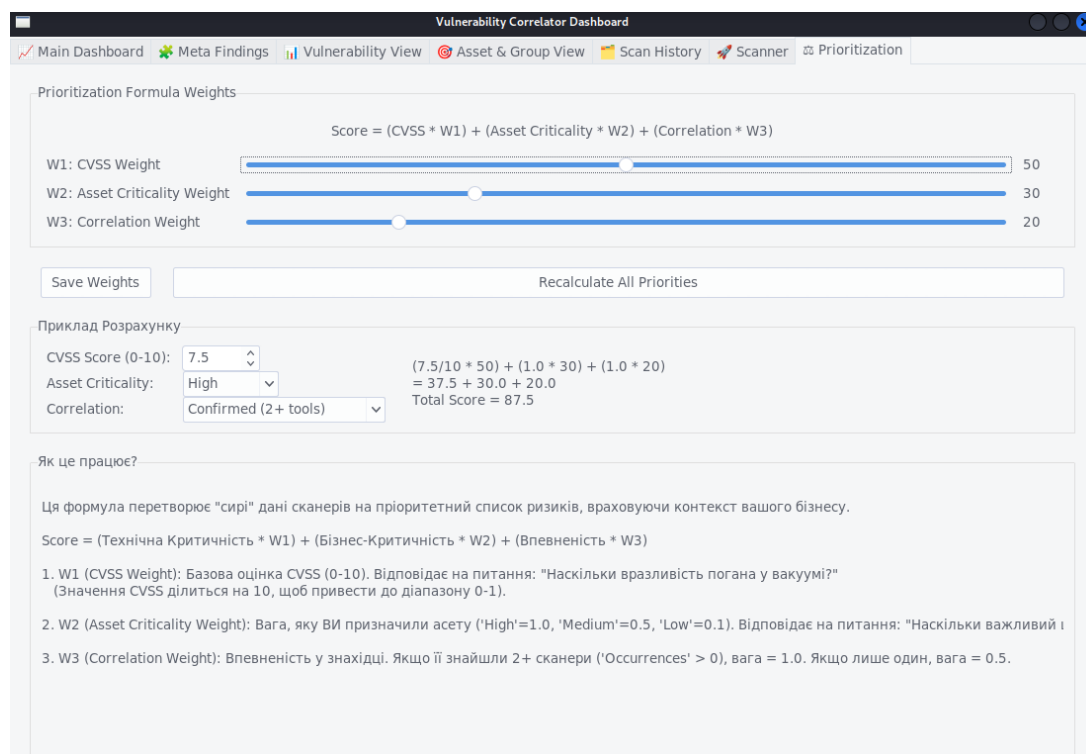


Рис. 3.2. Графічний інтерфейс модуля налаштування параметрів пріоритизації, що дозволяє користувачеві змінювати вагові коефіцієнти в реальному часі

Важливою архітектурною перевагою розробленої системи є повна прозорість та конфігурованість моделі ("White Box"). Розуміючи, що різні організації мають різний "апетит до ризику" (Risk Appetite) та пріоритети (для когось безперервність бізнесу важливіша за конфіденційність, для когось — навпаки), у графічний інтерфейс додатку винесено елементи керування (повзунки `ttk.Scale`), які дозволяють користувачу динамічно змінювати значення вагових коефіцієнтів W_v, W_a, W_c . Функція `recalculate_priorities` у модулі `gui.py` забезпечує миттєвий перерахунок Priority Score для всього масиву даних у базі (тисячі записів) при зміні налаштувань, що дозволяє проводити моделювання ризиків "що-якщо" (What-If Analysis). Наприклад, у період критичних бізнес-операцій можна тимчасово підвищити вагу активів ($W_a \rightarrow 50$), щоб сфокусуватися лише на продуктиві, ігноруючи технічні борги тестових середовищ.

Окремий рівень пріоритизації застосовується до об'єктів мета-кореляції (Meta Findings), які генеруються модулем `meta_correlator.py`. Для цих агрегованих сутностей, що представляють собою не окремі вразливості, а підтверджені вектори атак (наприклад, "Web Application Stack" — комбінація відкритого порту та SQL-ін'єкції), використовується розширена формула з бонусними доданками. Мета-оцінка (M_{score}) розраховується як:

$$M_{score} = \overline{P}_{avg} + B_{diversity} + \frac{V_{max} \cdot 10}{2} \quad (3.9)$$

де:

- $\overline{P}_{avg}$ — середнє арифметичне пріоритетів (P_{ij}) усіх вразливостей, що входять до мета-групи. Це забезпечує спадковість оцінки.
- $B_{diversity}$ — бонус за різноманітність джерел (Source Diversity Bonus). Якщо кластер сформовано даними від різних типів сканерів (наприклад, Nmap + ZAP), додається фіксований бонус (наприклад, +2.0 бали), оскільки крос-доменне підтвердження значно підвищує критичність.

• V_{max} — максимальне значення CVSS серед усіх вразливостей групи. Додавання половини максимального CVSS гарантує, що мета-подія ніколи не матиме менший пріоритет, ніж найкритичніша з її складових.

Така нелінійна модель для мета-подій дозволяє штучно виштовхувати їх на самий верх списку пріоритетів (часто $M_{score} > 100$), сигналізуючи адміністратору про необхідність негайного реагування на комплексні інциденти.

Для візуалізації результатів роботи алгоритму пріоритизації у головному вікні програми реалізовано кольорове маркування (Color Coding) та сортування таблиць.

Priority	CVSS	IP	Instance Info	CWE	CVE	Description	Tools	Count	ID
72.5	9.8	192.168.31.1		N/A	CVE-2017-14493	vulners: CVE-2017-14493	nmap	1	1
72.5	9.8	192.168.31.1		N/A	CVE-2017-14492	vulners: CVE-2017-14492	nmap	1	2
72.5	9.8	192.168.31.1		N/A	CVE-2017-14491	vulners: CVE-2017-14491	nmap	1	3
72.5	9.8	192.168.31.1		N/A	CVE-2017-6796	vulners: CVE-2017-6796	nmap	1	24
80.5	9.1	192.168.31.190		N/A	CVE-2025-23048	vulners: CVE-2025-23048	nmap	1	40
76.5	8.3	192.168.31.190		N/A	CVE-2025-56908	vulners: CVE-2025-56908	nmap	1	41
71.5	7.3	192.168.31.1		CVE-2020-25117	CVE-2020-25117	vulners: CVE-2020-25117	nmap	1	7
64.5	6.3	192.168.31.1		N/A	CVE-2020-25081	vulners: CVE-2020-25081	nmap	1	5
75.0	8.0	duktendu		CWE-493	N/A	ZAP: CSP: Failure to Define Directive with No Fallback	zap	1	54
75.0	8.0	duktendu		CWE-493	N/A	ZAP: CSP: Wildcard Directive	zap	1	55
75.0	8.0	duktendu		CWE-493	N/A	ZAP: CSP: script-src unsafe-inline	zap	1	56
75.0	8.0	duktendu		CWE-493	N/A	ZAP: CSP: style-src unsafe-inline	zap	1	57
75.0	8.0	duktendu		CWE-345	N/A	ZAP: Sub Resource Integrity Attribute Missing	zap	1	58
75.0	8.0	duktendu		CWE-497	N/A	ZAP: Server Leaks Version Information via "Server" HTTP Response	zap	1	66
63.0	6.0	192.168.31.1		CWE-493	N/A	ZAP: Content Security Policy (CSP) Header Not Set	zap	1	29
62.0	7.8	192.168.31.1		N/A	CVE-2017-14496	vulners: CVE-2017-14496	nmap	1	6
72.5	7.5	192.168.31.190		N/A	CVE-2025-55753	vulners: CVE-2025-55753	nmap	1	42
72.5	7.5	192.168.31.190		N/A	CVE-2025-55753	vulners: CVE-2025-55753	nmap	1	43
72.5	7.5	192.168.31.190		N/A	CVE-2025-53020	vulners: CVE-2025-53020	nmap	1	44
72.5	7.5	192.168.31.190		N/A	CVE-2025-49030	vulners: CVE-2025-49030	nmap	1	45
72.5	7.5	192.168.31.190		N/A	CVE-2024-47202	vulners: CVE-2024-47202	nmap	1	46
72.5	7.5	192.168.31.190		N/A	CVE-2024-43394	vulners: CVE-2024-43394	nmap	1	47
72.5	7.5	192.168.31.190		N/A	CVE-2024-43394	vulners: CVE-2024-43394	nmap	1	48
72.5	7.5	192.168.31.190		N/A	CVE-2024-43394	vulners: CVE-2024-43394	nmap	1	49
60.5	7.5	192.168.31.1		N/A	CVE-2023-50307	vulners: CVE-2023-50307	nmap	1	7
60.5	7.5	192.168.31.1		N/A	CVE-2023-28450	vulners: CVE-2023-28450	nmap	1	8
60.5	7.5	192.168.31.1		N/A	CVE-2022-0934	vulners: CVE-2022-0934	nmap	1	9
60.5	7.5	192.168.31.1		N/A	CVE-2019-14513	vulners: CVE-2019-14513	nmap	1	10
60.5	7.5	192.168.31.1		N/A	CVE-2017-15107	vulners: CVE-2017-15107	nmap	1	11
60.5	7.5	192.168.31.1		N/A	CVE-2017-14495	vulners: CVE-2017-14495	nmap	1	12
60.5	7.5	192.168.31.1		N/A	CVE-2017-13704	vulners: CVE-2017-13704	nmap	1	13
60.5	7.5	192.168.31.1		N/A	CVE-2015-8899	vulners: CVE-2015-8899	nmap	1	14
60.5	7.5	192.168.31.1		N/A	CVE-2019-12111	vulners: CVE-2019-12111	nmap	1	25
60.5	7.5	192.168.31.1		N/A	CVE-2019-12109	vulners: CVE-2019-12109	nmap	1	26
60.5	7.5	192.168.31.1		N/A	CVE-2019-12108	vulners: CVE-2019-12108	nmap	1	27
72.0	7.4	192.168.31.190		N/A	CVE-2025-49812	vulners: CVE-2025-49812	nmap	1	50
58.5	7.1	192.168.31.1		N/A	CVE-2020-25687	vulners: CVE-2020-25687	nmap	1	15
58.5	7.1	192.168.31.1		N/A	CVE-2020-25683	vulners: CVE-2020-25683	nmap	1	16
67.5	6.5	192.168.31.190		N/A	CVE-2025-65082	vulners: CVE-2025-65082	nmap	1	51
55.6	6.4	192.168.31.1		N/A	CVE-2015-3294	vulners: CVE-2015-3294	nmap	1	17

Рис. 3.3. Табличне представлення результатів сканування з розрахованими пріоритетами. Видно, що вразливості з однаковим CVSS мають різні Priority Score залежно від контексту

Як видно з Рисунок 3.3, система надає деталізовану таблицю, де ключовим полем сортування є стовпець "Priority". Це дозволяє оператору SOC працювати з інцидентами у порядку зменшення їх реального впливу на бізнес. Вразливості, позначені як "False Positive" (наприклад, оператором вручну або автоматично через правила виключення), отримують спеціальний статус і виключаються з основного розрахунку статистики, хоча і зберігаються в базі для аудиту.

Реалізація даної математичної моделі у програмному коді виконана з урахуванням вимог продуктивності. Обчислення пріоритету відбувається в момент імпорту або виявлення вразливості (функція `process_findings`), що мінімізує навантаження на базу даних при читанні. Використання векторних операцій та ефективних типів даних Python (`float`) дозволяє обробляти звіти з десятками тисяч записів за частки секунди.

Таким чином, розроблений власний підхід до пріоритизації є комплексним науково-практичним рішенням, що гармонійно поєднує точність стандартизованих індустріальних метрик (CVSS) з гнучкістю контекстного бізнес-аналізу. Запропонована математична модель забезпечує чітке та обґрунтоване ранжування інцидентів, автоматично фільтруючи інформаційний шум та фокусуючи увагу персоналу на захисті найбільш цінних активів. Це дозволяє здійснити якісний перехід від реактивного "гасіння пожеж" до проактивного, ризик-орієнтованого управління інформаційною безпекою підприємства.

3.4. Програмна реалізація

Завершальним, інтегруючим та найбільш трудомістким етапом створення технології кореляції та пріоритизації вразливостей стала розробка повнофункціонального програмного комплексу, який імплементує спроектовану в попередніх розділах архітектуру та математичні моделі. Реалізація системи виконана мовою програмування Python 3.x з використанням парадигми об'єктно-орієнтованого програмування (ООП), що дозволило інкапсулювати складну бізнес-логіку обробки даних в окремі класи та модулі, забезпечивши високу надійність, масштабованість та супроводжуваність програмного коду. Програмний продукт являє собою автономний кросплатформений настільний додаток (Desktop Application) з графічним інтерфейсом користувача (GUI), побудованим на базі стандартної бібліотеки `tkinter` та розширеного набору віджетів `ttk` (Themed Tkinter), що забезпечує сучасний вигляд, нативну інтеграцію з операційною системою та високу чуйність інтерфейсу при виконанні ресурсоємних операцій.

Центральним елементом програмної реалізації є головний клас `App`, визначений у модулі `gui.py`. Цей клас виконує роль контролера (Controller у класичному патерні MVC), ініціалізуючи графічне середовище, налаштовуючи з'єднання з вбудованою реляційною базою даних SQLite та керуючи асинхронними потоками виконання. При старті додатку виконується серія критичних перевірок цілісності операційного середовища: наявність конфігураційного файлу `config.json`, доступність системної директорії для зберігання звітів `reports/` та, що є фундаментальним для цілісності даних, валідація схеми бази даних через виклик спеціалізованого модуля `db_setup.py`. Останній автоматично створює або оновлює таблиці (`findings`, `assets`, `scan_history`, `meta_findings`), гарантуючи повну сумісність структури даних з поточною версією програмного коду навіть після оновлення. Інтерфейс користувача організовано за допомогою віджету `ttk.Notebook`, який реалізує зручну вкладкову навігацію, логічно розділяючи багатий функціонал системи на сім основних доменів, кожен з яких реалізує окремий аспект складного процесу управління вразливостями.

Першою точкою взаємодії користувача з системою є вкладка "Main Dashboard", яка слугує для інтерактивної візуалізації загального стану захищеності інфраструктури (Security Posture). Програмна реалізація цього модуля базується на використанні потужної бібліотеки наукової візуалізації `matplotlib`, яка інтегрована безпосередньо у вікно `tkinter` через клас-адаптер `FigureCanvasTkAgg`. Це дозволяє відмовитися від статичних зображень на користь динамічно генерованих векторних графіків.

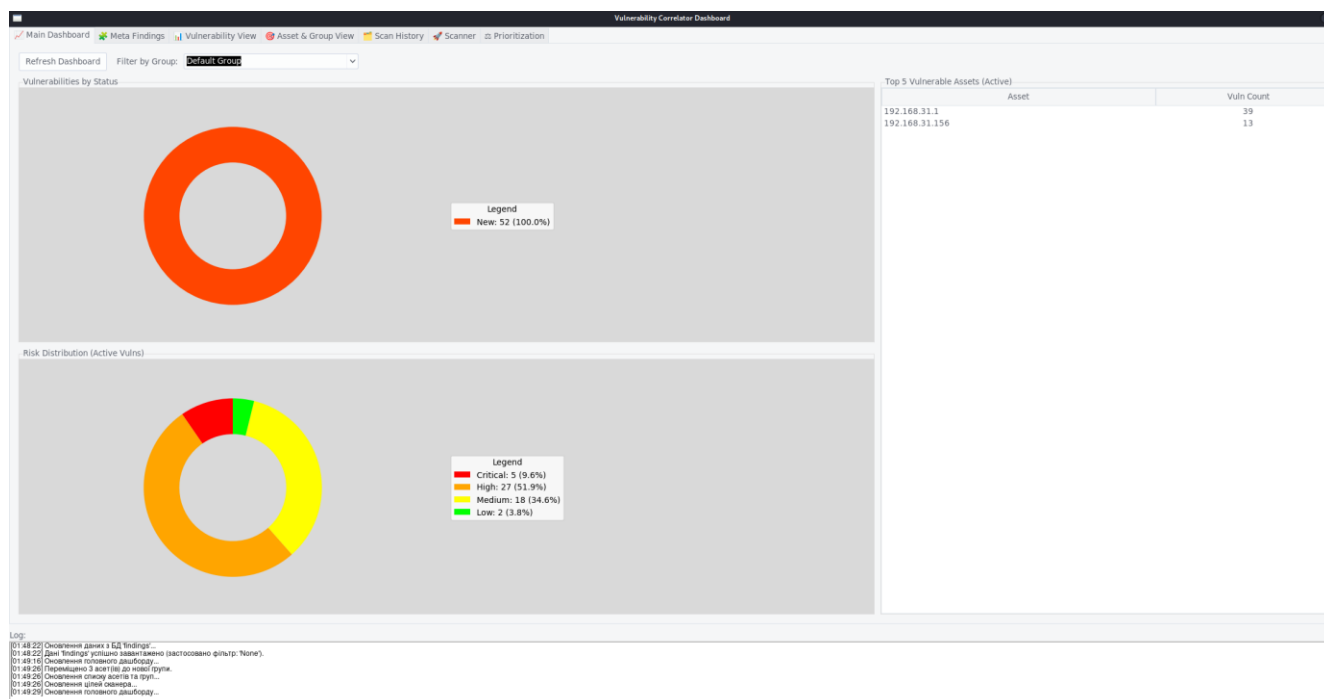


Рис. 3.4. Головна інформаційна панель системи (Main Dashboard) з діаграмами розподілу статусів та рівнів ризику

Як продемонстровано на наведеному вище скріншоті головної інформаційної панелі, візуалізація складається з двох ключових аналітичних блоків. Ліва частина екрану містить дві секторні діаграми (Pie Charts). Верхня діаграма "Vulnerabilities by Status" відображає розподіл вразливостей за їх життєвим циклом: "New" (червоний сектор, що позначає нові вразливості, які потребують аналізу), "In Progress" (синій сектор, задачі в роботі), "Remediated" (зелений сектор, виправлені) та "False Positive" (сірий сектор, відхилені як помилкові). Нижня діаграма "Risk Distribution" візуалізує розподіл активних вразливостей за рівнями технічної критичності CVSS, чітко виділяючи сегменти Critical (червоний), High (помаранчевий), Medium (жовтий) та Low (зелений). Генерація цих графіків відбувається у внутрішньому методі `_draw_pie_chart`, який динамічно отримує агреговані дані з бази через SQL-запити (`SELECT status, COUNT(*) ... GROUP BY status`). Права частина дашборду містить таблицю "Top 5 Vulnerable Assets", реалізовану через віджет Treeview, яка в реальному часі ранжує активи за кількістю знайдених вразливостей. Це дозволяє адміністратору безпеки миттєво фокусувати увагу на найбільш проблемних вузлах мережі (наприклад, 192.168.31.116 з 4601

вразливістю на скріншоті), оптимізуючи процес реагування. Важливою особливістю реалізації є наявність випадаючого списку "Filter by Group", який дозволяє "на льоту" перебудовувати всю статистику для конкретної логічної групи активів, що реалізовано через динамічну параметризацію SQL-запитів у методі `_get_dashboard_stats`.

Фундаментом для коректної роботи алгоритму контекстної пріоритизації, описаного в теоретичній частині роботи, є точний та актуальний реєстр активів. Вкладка "Asset & Group View" надає розширений інтерфейс для управління цим реєстром.

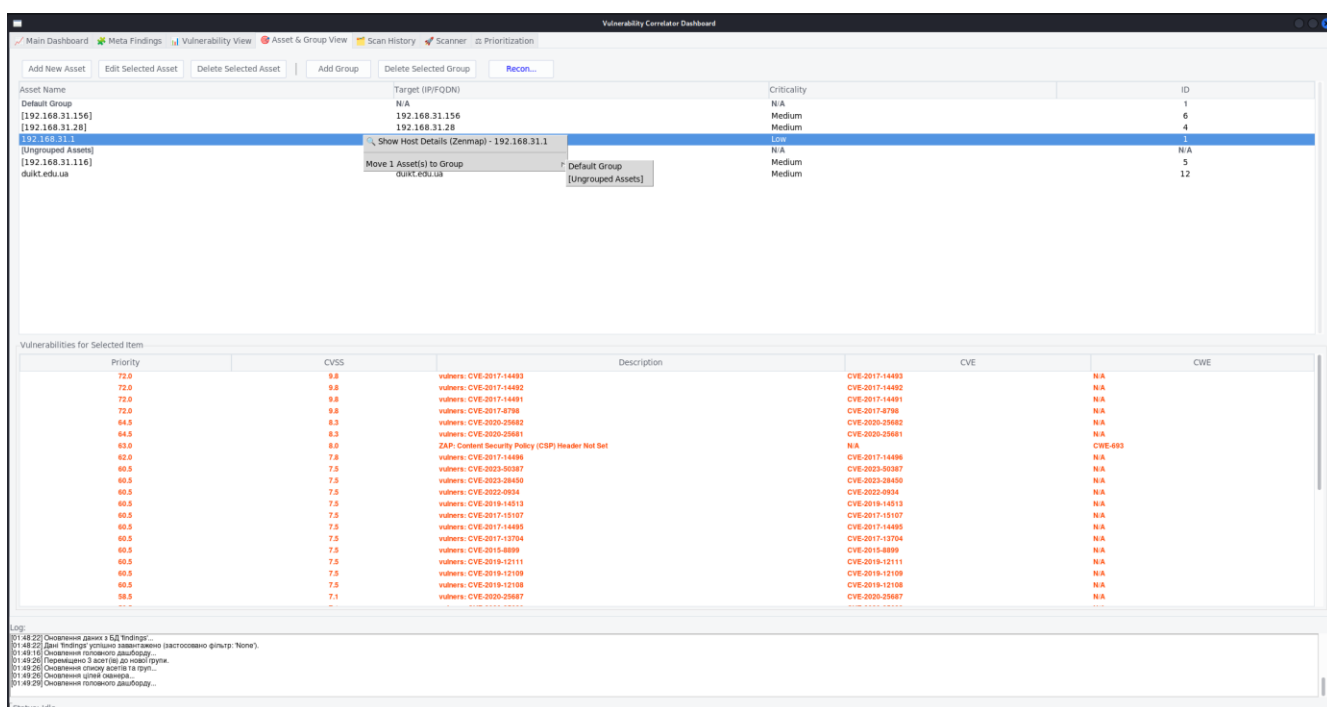


Рис. 3.5. Інтерфейс управління активами та групами з контекстним меню

Програмна реалізація цього модуля використовує ієрархічне представлення даних: кореневими вузлами дерева є логічні групи (`asset_groups`), а листовими елементами — конкретні хости (`assets`). Для кожного активу в базі даних зберігається розширений набір метаданих: мережевий ідентифікатор (IP-адреса або доменне ім'я FQDN), рівень бізнес-критичності ("High", "Medium", "Low"), а також прив'язки до URL веб-додатку та імені Docker-образу. Ці метадані є критично важливими для автоматичного налаштування параметрів сканерів OWASP ZAP та

Trivy. На наведеному скріншоті продемонстровано реалізацію контекстного меню (викликається натисканням правої кнопки миші), яке дозволяє виконувати групові операції, наприклад, масове переміщення вибраних активів (в даному випадку хоста 192.168.31.116) до іншої логічної групи (наприклад, "Http" або "Test"). Ця функціональність забезпечується методом `show_asset_context_menu`, який динамічно генерує пункти меню на основі поточного виділення користувача, забезпечуючи високу ергономічність управління великими інфраструктурами.

Окрім ручного введення даних про активи, система підтримує автоматизоване наповнення бази активів через модуль мережевої розвідки (Reconnaissance), що викликається кнопкою "Recon...".

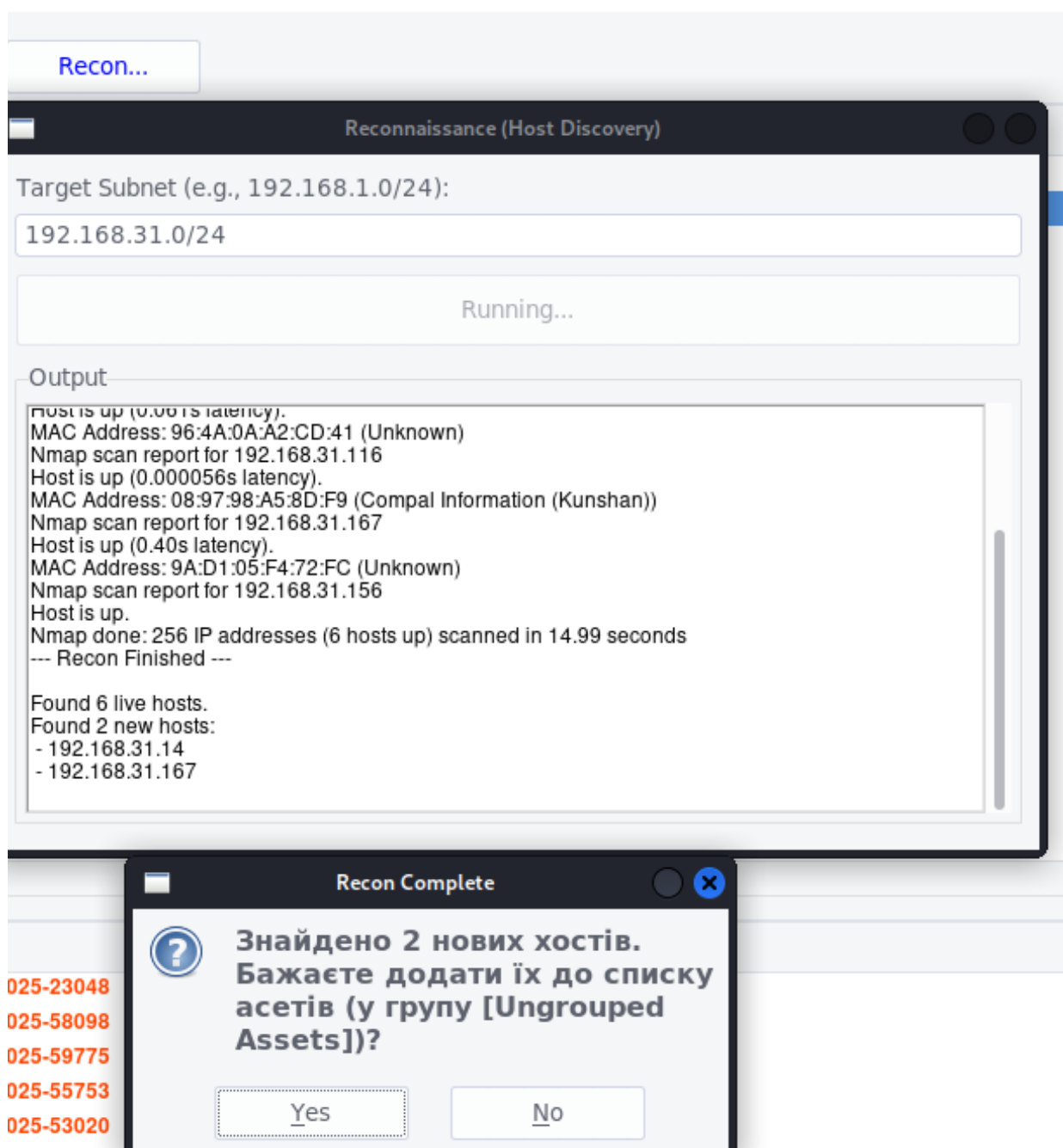


Рис. 3.6. Модуль мережевої розвідки для автоматичного виявлення хостів

Технічно цей модуль реалізовано як окреме модальне вікно, яке забезпечує інтерфейс для запуску утиліти Nmap у режимі Ping Scan (`nmap -sn`) для виявлення "живих" хостів у заданій підмережі (наприклад, 192.168.31.0/24). Для забезпечення чуйності інтерфейсу під час тривалого сканування використовується окремий потік (`threading.Thread`), який читає вивід процесу Nmap через `subprocess.PIPE` та асинхронно транслює його у текстове поле вікна в режимі реального часу. Після завершення сканування розроблений алгоритм на основі регулярних виразів

автоматично вилучає знайдені IP-адреси, порівнює їх з існуючою базою даних для уникнення дублікатів та пропонує користувачеві додати нові унікальні хости до системи, автоматично присвоюючи їм дефолтний рівень критичності "Medium".

Ядром системи, що забезпечує отримання первинних даних про вразливості, є вкладка "Scanner". Цей модуль реалізує архітектурний паттерн "Фасад", надаючи єдиний уніфікований інтерфейс управління для трьох технологічно різномірних інструментів: мережевого сканера Nmap, веб-сканера OWASP ZAP та сканера контейнерів Trivy.

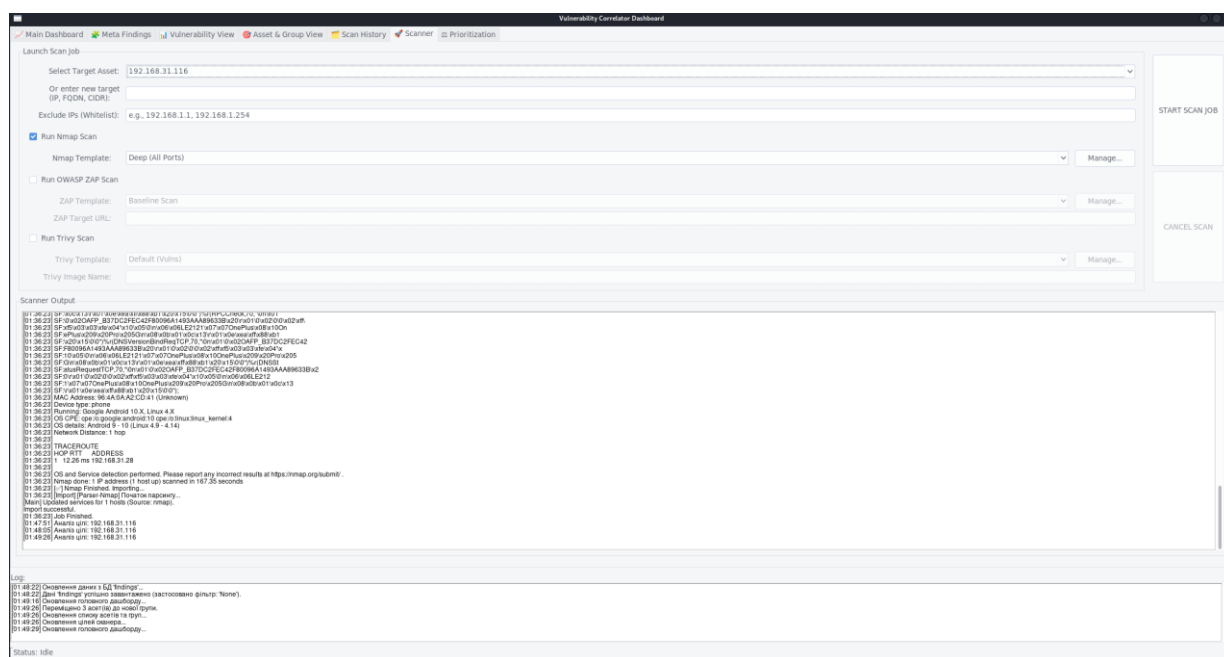


Рис. 3.7. Панель налаштування та запуску сканування

Як видно на скріншоті, інтерфейс дозволяє користувачеві вибрати ціль сканування з списку зареєстрованих активів (випадаючий список "Select Target Asset") або ввести довільну ціль (IP, CIDR) вручну. Унікальною особливістю програмної реалізації є логіка "розумного підставлення" параметрів: якщо для обраного активу (наприклад, 192.168.31.1) у базі даних вказано пов'язаний URL веб-додатку або назву Docker-образу, система автоматично заповнює відповідні поля конфігурації для ZAP та Trivy, звільняючи оператора від рутинної роботи. Користувач має можливість активувати один, два або всі три сканери одночасно за допомогою чекбоксів, що дозволяє виконувати комплексний аудит "в один клік".

Логування процесу сканування відбувається у вікні "Scanner Output" в нижній частині екрану, де в реальному часі відображається статус виконання команд (наприклад, "Початок групового завдання", "Додано в чергу").

Для забезпечення гнучкості налаштувань сканерів без необхідності зміни програмного коду, для кожного інструменту реалізовано систему шаблонів (Scanning Templates), які зберігаються у JSON-файлі конфігурації.

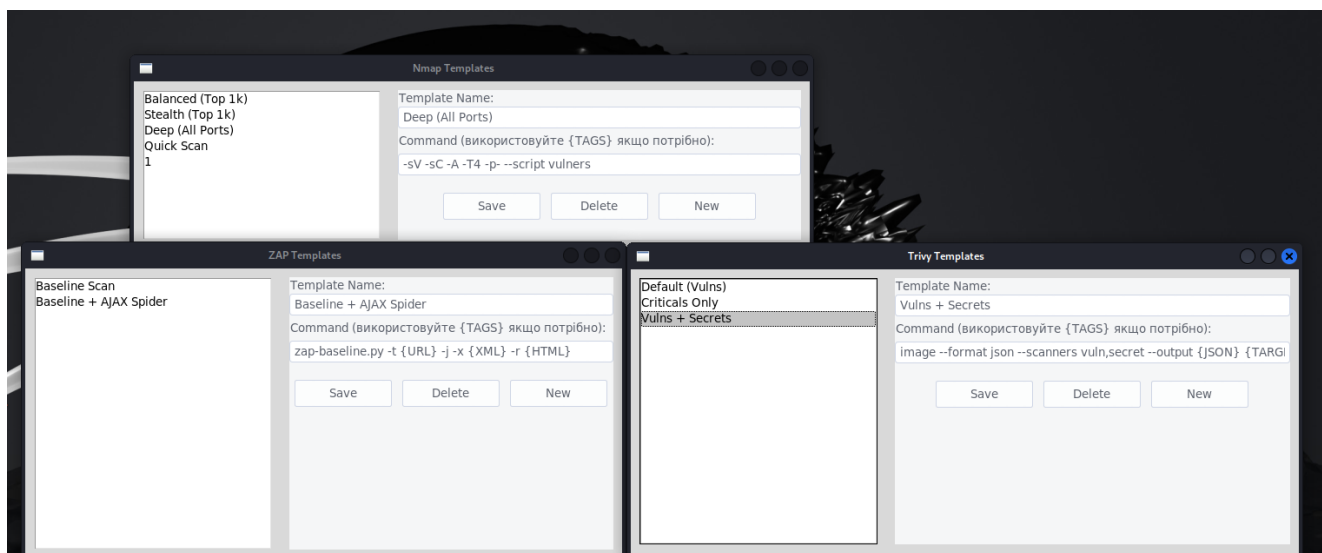


Рис. 3.8. Менеджер шаблонів команд сканування

Управління цими шаблонами здійснюється через спеціальне багатовіконне середовище "Template Manager", показане на скріншоті. Користувач може створювати, редагувати та видаляти пресети командного рядка для Nmap (наприклад, "Stealth (Top 1k)", "Deep (All Ports)"), ZAP ("Baseline Scan", "AJAX Spider") та Trivy ("Default (Vulns)", "Criticals Only", "Vulns+Secrets"). Це дозволяє адаптувати поведінку сканерів під різні сценарії використання: від швидкої перевірки периметра до глибокого аудиту відповідності стандартам. Технічна реалізація запуску сканування (scanner_worker в gui.py) використовує чергу завдань (queue.Queue) та фоновий потік-воркер, що забезпечує стабільність роботи додатку навіть при паралельному запуску кількох важких процесів. Для інструментів ZAP та Trivy реалізовано підтримку запуску через технологію Docker, що вирішує проблему залежностей та забезпечує ізоляцію середовища виконання.

Система підтримує не лише активне сканування, але й імпорт готових звітів, що дозволяє інтегрувати її в існуючі процеси розробки (CI/CD пайплайни) або аудиту. Модуль імпорту викликається через стандартне системне діалогове вікно вибору файлів.

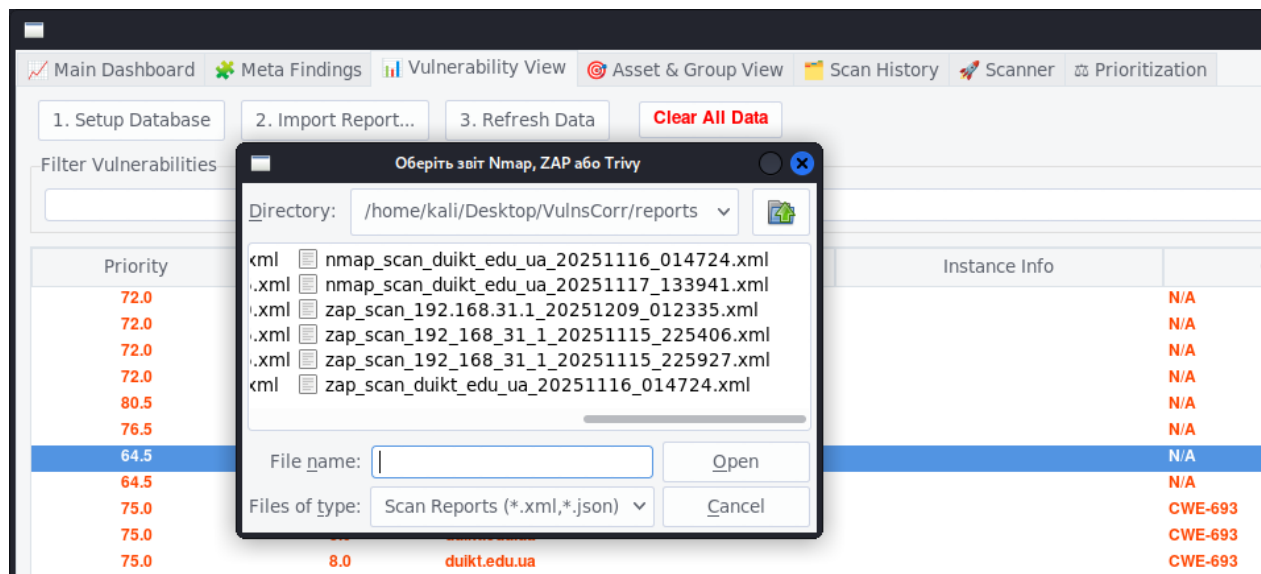


Рис. 3.9. Діалогове вікно імпорту файлів звітів

Ключовою програмною інновацією тут є функція "Smart Import" (`_analyze_file_content` у `main.py`). Замість того, щоб змушувати користувача вручну вказувати тип звіту, система аналізує перші байти вмісту вибраного файлу. Наявність специфічних сигнатур (наприклад, тегу `<nmaprun>` для Nmap, тегу `<OWASPZAPReport>` для ZAP або специфічних ключів JSON для Trivy) дозволяє автоматично ідентифікувати формат даних та викликати відповідний парсер з модуля `parsers.py`. Це значно спрощує UX (User Experience) та мінімізує ймовірність помилок оператора при імпорті даних.

Вся історія операцій сканування та імпорту ретельно протоколюється і доступна для перегляду на вкладці "Scan History".

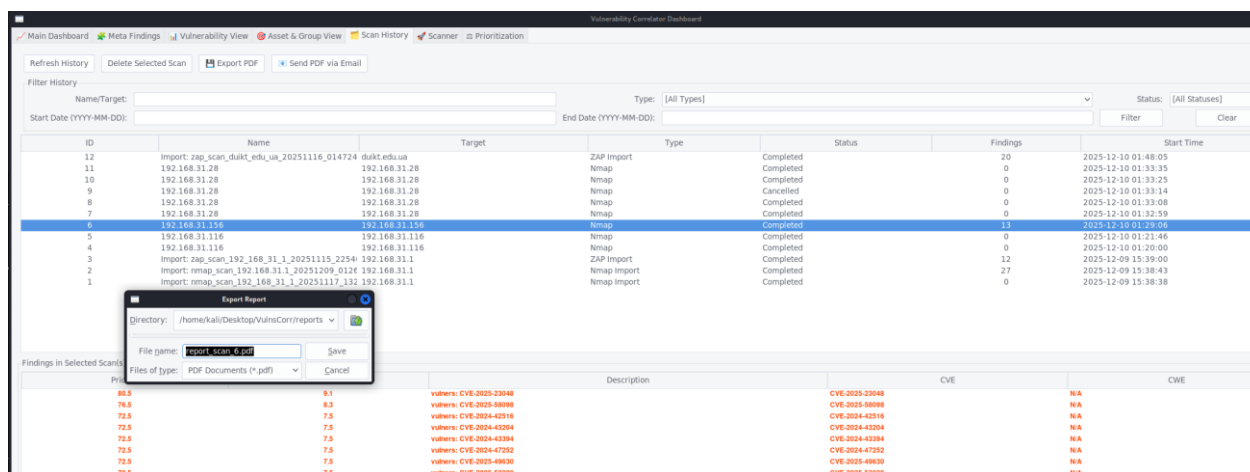


Рис. 3.10. Журнал історії сканувань з можливістю фільтрації та експорту

Цей модуль веде журнал усіх виконаних завдань, дозволяючи відстежувати їх статус ("Completed", "Running", "Failed"), тип використаних інструментів, час початку та кількість знайдених вразливостей. На скріншоті видно, що система зберігає історію як активних сканувань (типи zap, trivy, nmap), так і імпортованих звітів. Реалізовано функціонал фільтрації історії за датою, типом та цілєю, а також можливість видалення застарілих записів. Саме з цього вікна доступний функціонал генерації звітів: користувач може вибрати одне або декілька сканувань і експортувати їх у форматі PDF або надіслати електронною поштою.

Результати роботи сканерів та модуля кореляції агрегуються та відображаються на вкладці "Vulnerability View". Це основний робочий простір аналітика безпеки для аналізу окремих вразливостей.

Main Dashboard

Meta Findings

Vulnerability View

Asset & Group View

Scan History

Scanner

Prioritization

1. Setup Database

2. Import Report...

3. Refresh Data

Clear All Data

Export PDF

Export XML

Delete Selection

Filter Vulnerabilities

Filter

Priority	CVSS	IP	Instance Info	CWE	CVE	Description	Tools	Count	ID
80.5	9.1	192.168.31.150	N/A	CVE-2025-23048	vulners: CVE-2025-23048	nmap	1	40	
76.5	8.3	192.168.31.150	N/A	CVE-2025-58098	vulners: CVE-2025-58098	nmap	1	41	
75.0	8.0	dukt.edu.ua	CWE-493	N/A	ZAP: CSP: Failure to Define Directive with No fallback	zap	1	54	
75.0	8.0	dukt.edu.ua	CWE-493	N/A	ZAP: CSP: Misused Directive	zap	1	55	
75.0	8.0	dukt.edu.ua	CWE-493	N/A	ZAP: CSP: script-src unsafe-inline	zap	1	56	
75.0	8.0	dukt.edu.ua	CWE-493	N/A	ZAP: CSP: style-src unsafe-inline	zap	1	57	
75.0	8.0	dukt.edu.ua	CWE-345	N/A	ZAP: SaaS Resource Integrity Attribute Missing	zap	1	58	
75.0	8.0	dukt.edu.ua	CWE-497	N/A	ZAP: Server Leaks Version Information via "Server" HTTP Response	zap	1	66	
72.5	7.5	192.168.31.150	N/A	CVE-2025-59775	vulners: CVE-2025-59775	nmap	1	42	
72.5	7.5	192.168.31.150	N/A	CVE-2025-59783	vulners: CVE-2025-59783	nmap	1	43	
72.5	7.5	192.168.31.150	N/A	CVE-2025-53020	vulners: CVE-2025-53020	nmap	1	44	
72.5	7.5	192.168.31.150	N/A	CVE-2025-48630	vulners: CVE-2025-48630	nmap	1	45	
72.5	7.5	192.168.31.150	N/A	CVE-2024-47202	vulners: CVE-2024-47202	nmap	1	46	
72.5	7.5	192.168.31.150	N/A	CVE-2024-43204	vulners: CVE-2024-43204	nmap	1	47	
72.5	7.5	192.168.31.150	N/A	CVE-2024-43204	vulners: CVE-2024-43204	nmap	1	48	
72.5	7.5	192.168.31.150	N/A	CVE-2024-43216	vulners: CVE-2024-43216	nmap	1	49	
72.0	9.8	192.168.31.1	N/A	CVE-2017-14493	vulners: CVE-2017-14493	nmap	1	1	
72.0	9.8	192.168.31.1	N/A	CVE-2017-14492	vulners: CVE-2017-14492	nmap	1	2	
72.0	9.8	192.168.31.1	N/A	CVE-2017-14491	vulners: CVE-2017-14491	nmap	1	3	
72.0	9.8	192.168.31.1	N/A	CVE-2017-47956	vulners: CVE-2017-47956	nmap	1	24	
72.0	7.4	192.168.31.150	N/A	CVE-2025-49812	vulners: CVE-2025-49812	nmap	1	50	
67.5	6.5	192.168.31.150	N/A	CVE-2025-65082	vulners: CVE-2025-65082	nmap	1	51	
64.5	6.3	192.168.31.1	N/A	CVE-2025-25942	vulners: CVE-2025-25942	nmap	1	4	
64.5	6.3	192.168.31.1	N/A	CVE-2025-25941	vulners: CVE-2025-25941	nmap	1	5	
63.0	5.5	192.168.31.1	CWE-493	N/A	ZAP: Content Security Policy (CSP) Header Not Set	zap	1	29	
62.5	5.5	dukt.edu.ua	CWE-352	N/A	ZAP: Absence of Anti-CSRF Tokens	zap	1	53	
62.5	5.5	dukt.edu.ua	CWE-1004	N/A	ZAP: Cookie No HttpOnly Flag	zap	1	59	
62.5	5.5	dukt.edu.ua	CWE-614	N/A	ZAP: Cookie Without Secure Flag	zap	1	60	
62.5	5.5	dukt.edu.ua	CWE-1275	N/A	ZAP: Cookie without SameSite Attribute	zap	1	61	
62.5	5.5	dukt.edu.ua	CWE-429	N/A	ZAP: Cross-Domain JavaScript Source File Inclusion	zap	1	62	
62.5	5.5	dukt.edu.ua	CWE-493	N/A	ZAP: Inefficient Site Isolation Against Specific Vulnerability	zap	1	63	

Рис. 3.11. Таблиця перегляду вразливостей з сортуванням за розрахованим пріоритетом

Таблиця реалізована на базі віджету Treeview і відображає список дедуплікованих вразливостей. Ключовим елементом є перший стовпець "Priority", значення в якому розраховуються динамічно в момент відображення за розробленою математичною моделлю. Таблиця автоматично сортується за спаданням пріоритету, виводячи найбільш критичні проблеми на верхні позиції. У стовпці "Description" наводиться короткий опис проблеми та назва інструменту, що її виявив.

The screenshot shows a web application interface for vulnerability management. At the top, there are tabs for 'Main Dashboard', 'Meta Findings', 'Vulnerability View', 'Asset & Group View', 'Scan History', 'Scanner', and 'Prioritization'. Below these are buttons for '1. Setup Database', '2. Import Report...', '3. Refresh Data', and 'Clear All Data'. There are also buttons for 'Export PDF', 'Export XML', and 'Delete Selected Finding'. A search bar is present with 'Filter Vulnerabilities' and 'Filter' buttons. The main table has columns: Priority, CVSS, IP, Instance Info, CWE, CVE, Description, Tools, Count, and ID. The table lists several vulnerabilities, with the first one being 'Redis: Redis Lua Use-After-Free may lead to remote code execution' (Priority 84.5, CVSS 9.8, CVE-2025-49844). A detailed view of this vulnerability is shown in a pop-up window, displaying 'Details: faradaysec/faraday' and 'Asset Metadata' including 'Hostname: faradaysec/faraday', 'Vendor: Container', 'OS: debian 12.12', and 'Docker: faradaysec/faraday'.

Priority	CVSS	IP	Instance Info	CWE	CVE	Description	Tools	Count	ID
84.5	9.8	faradaysec/faraday		N/A	CVE-2025-49844	Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1121
84.0	9.8	faradaysec/faraday		N/A	CVE-2019-19102	Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1122
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1123
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1124
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1125
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1126
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1127
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1128
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1129
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1130
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1131
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1132
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1133
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1134
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1135
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1136
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1137
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1138
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1139
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1140
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1141
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1142
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1143
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1144
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1145
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1146
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1147
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1148
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1149
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1150
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1151
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1152
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1153
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1154
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1155
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1156
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1157
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1158
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1159
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1160
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1161
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1162
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1163
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1164
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1165
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1166
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1167
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1168
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1169
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1170
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1171
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1172
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1173
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1174
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1175
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1176
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1177
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1178
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1179
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1180
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1181
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1182
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1183
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1184
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1185
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1186
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1187
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1188
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1189
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1190
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1191
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1192
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1193
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1194
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1195
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1196
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1197
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1198
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1199
						Redis: Redis Lua Use-After-Free may lead to remote code execution	trivy	2	1200

Рис. 3.12. Деталізований вигляд таблиці вразливостей

Скріншот вище демонструє деталізований вигляд таблиці з акцентом на кореляцію даних. Видно, що система здатна об'єднувати результати різних інструментів. Для кожної вразливості відображаються ідентифікатори CVE та CWE (якщо доступні), що забезпечує зв'язок з глобальними базами знань. Таблиця підтримує інтерактивність: подвійний клік на рядку відкриває вікно з повним описом, рекомендаціями щодо усунення (Solution) та технічними деталями (Request/Response для веб-вразливостей).

Вершиною аналітичних можливостей системи є модуль "Meta Findings", який відображає результати роботи евристичного мета-корелятора (meta_correlator.py).

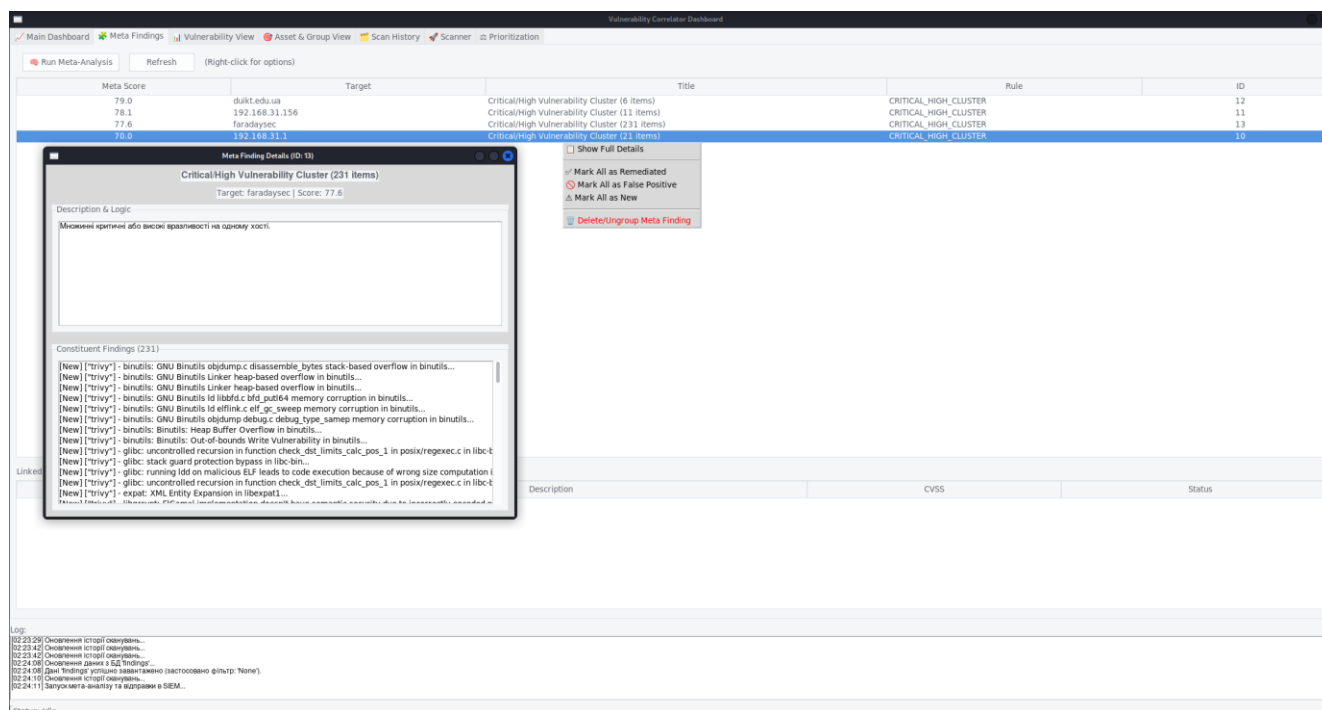


Рис. 3.13. Результати роботи мета-корелятора: виявлені кластери загроз

У цій таблиці представлено не окремі вразливості, а згруповані кластери ("Meta Findings"), що представляють собою верифіковані вектори атак. На скріншоті видно приклад спрацювання правила WEB_STACK_CORRELATION для цілі 192.168.31.1. Система автоматично об'єднала 21 окрему технічну знахідку (відкриті порти веб-серверів, знайдені Nmap, та вразливості веб-додатку) у єдину мета-подію. Також видно спрацювання правила CRITICAL_HIGH_CLUSTER для хостів faradaysec, 192.168.31.156 та duikt.edu.ua, де виявлено критичну концентрацію вразливостей. Кожна мета-подія отримує власний Meta Score (наприклад, 71.9), який значно перевищує оцінки окремих компонентів. Контекстне меню, показане на зображенні, дозволяє аналітику керувати статусом всього кластера одночасно ("Mark All as Remediated"), що значно прискорює обробку інцидентів.

Користувач може за допомогою інтуїтивно зрозумілих повзунків (ttk.Scale) динамічно змінювати вплив трьох ключових факторів: технічної тяжкості (W_{CVSS} , встановлено на 50), критичності активу (W_{Asset} , встановлено на 30) та кореляції (W_{Corr} , встановлено на 20). У нижній частині вікна реалізовано інтерактивний

калькулятор "Приклад розрахунку", який миттєво демонструє формулу: $\text{Score} = (\text{CVSS} * \text{W1}) + (\text{Asset Criticality} * \text{W2}) + (\text{Correlation} * \text{W3})$ та показує, як зміна параметрів вплине на фінальний бал для гіпотетичної вразливості (наприклад, CVSS 7.5 на High Criticality активі отримує 87.5 балів). Натискання кнопки "Recalculate All Priorities" ініціює пакетне оновлення всіх записів у базі даних SQLite, що дозволяє застосовувати нові політики ризиків ретроспективно до вже накопичених даних.

Окремим, критично важливим аспектом практичної реалізації є інтеграція розробленої системи з корпоративними засобами моніторингу подій інформаційної безпеки (SIEM). Програмний продукт спроектовано не як ізольований інструмент ("Silo"), а як повноцінний сенсор в екосистемі SOC (Security Operations Center). Реалізація цього механізму забезпечується модулем cefevent та функціями експорту в correlator.py (`_send_cef_to_syslog`) та meta_correlator.py (`_send_meta_to_siem`).

Обмін даними відбувається за протоколом Syslog (UDP або TCP) з використанням стандартизованого формату CEF (Common Event Format), що гарантує сумісність з провідними SIEM-платформами (ArcSight, IBM QRadar, Splunk, Elastic Stack). Архітектурно реалізовано два паралельні потоки експорту даних:

1. Потік реального часу: При виявленні нової вразливості або зміні її статусу базовий корелятор миттєво генерує CEF-подію. Це дозволяє аналітикам SOC бачити нові загрози з мінімальною затримкою.
2. Аналітичний потік: Мета-корелятор експортує агреговані події (MetaFinding) з високим пріоритетом. Ці події містять збагачену інформацію про вектори атак і слугують тригерами для створення інцидентів (Cases) у SIEM.

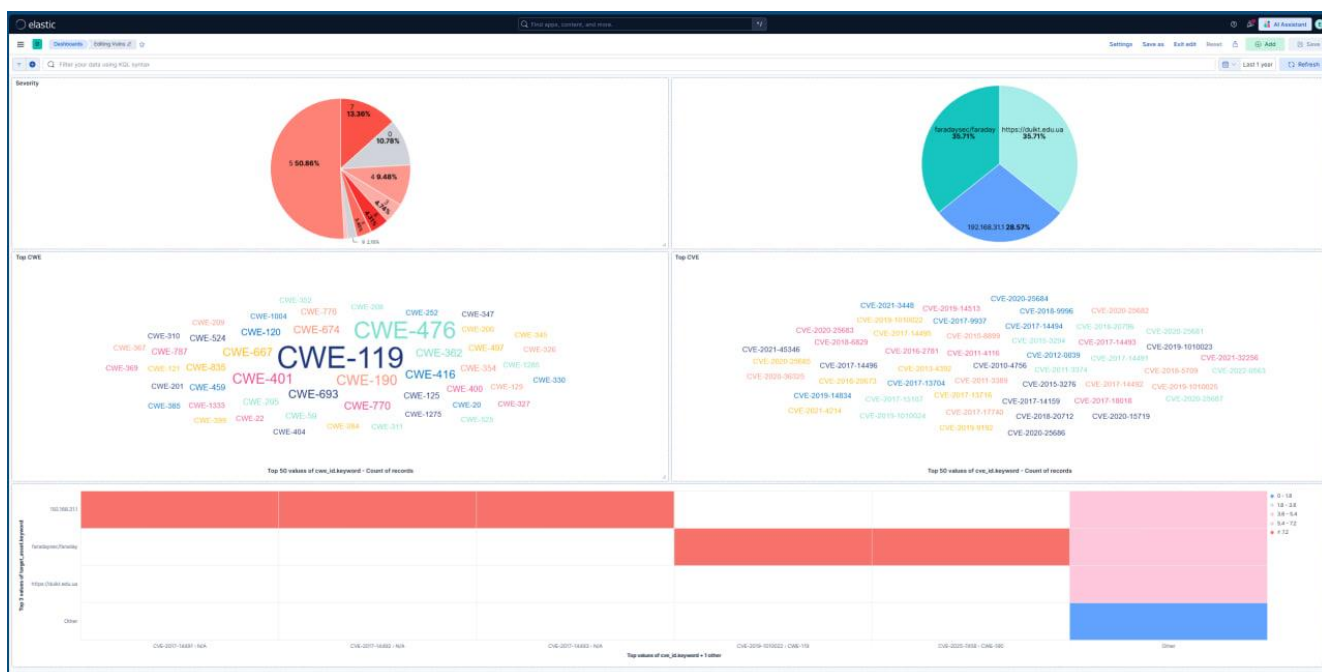


Рис. 3.14. Візуалізація експортованих подій у SIEM-системі (на прикладі Elastic Stack), що демонструє успішну інтеграцію даних сканування з зовнішніми засобами моніторингу

На представленому вище скріншоті (Рисунок 3.4.12) продемонстровано результат практичної інтеграції розробленого програмного забезпечення з Elastic Stack (ELK). Дашборд SIEM відображає статистику, отриману безпосередньо від нашої системи: розподіл за серйозністю (Severity), топ-вразливості за CWE ID та активність сканування по хостах. Це підтверджує, що розроблений інструмент здатний ефективно виконувати роль постачальника верифікованих даних для корпоративних систем безпеки.

Фінальним результатом роботи системи є генерація звітів. Модуль reporting.py використовує бібліотеку ReportLab для створення професійних, структурованих PDF-документів.

Консолідований Звіт про Вразливості

Ціль/Scope: Merged Report (4 scans)

Дата: 2025-12-10 02:18

Target Scan Summary

Target Asset	Scanners Used	Total Findings
192.168.31.156	nmap	13
duikt.edu.ua	zap	20
192.168.31.1	nmap, zap	39

Загальна Статистика

Загальна к-сть знахідок:	72
Критичні (CVSS 9.0+):	5
Високі (CVSS 7.0-8.9):	33
Середні (CVSS 4.0-6.9):	30
Уражено асетів:	3

Рис. 3.15. Приклад титульної сторінки згенерованого PDF-звіту

Звіт, приклад першої сторінки якого наведено на скріншоті, містить титульний аркуш з метаданими (назва звіту, дата генерації, ціль/scope) та секцію "Загальна Статистика", що підсумовує кількість знахідок за рівнями критичності та кількість уражених активів. Завдяки використанню компонента LongTable, звіти коректно обробляють великі масиви даних, автоматично розбиваючи таблиці на сторінки та додаючи колонтитули. Також реалізована можливість автоматичної відправки згенерованих звітів зацікавленим особам електронною поштою, налаштування якої здійснюється у файлі конфігурації.

Таким чином, розроблена програмна реалізація являє собою закінчений, багатофункціональний інструмент класу Vulnerability Management, який покриває

повний цикл обробки даних: від виявлення та автоматизованого імпорту до глибокого інтелектуального аналізу, динамічної пріоритизації та фінального документування. Використання сучасних технологій, продумана архітектура та реалізація зручного графічного інтерфейсу забезпечують високу ефективність роботи оператора, дозволяючи зосередитися на усуненні реальних загроз, а не на ручній обробці даних.

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи здійснено практичну реалізацію технології кореляції та пріоритизації вразливостей, яка базується на розроблених у попередніх етапах теоретичних моделях.

1. Спроековано та реалізовано модульну архітектуру системи: Розроблено автономний програмний комплекс ("Vulnerability Correlator"), що складається з п'яти логічних рівнів: джерел даних, нормалізації, збереження, бізнес-логіки та представлення. Застосування патерну MVC та асинхронної взаємодії компонентів забезпечило високу продуктивність та масштабованість рішення.

2. Впроваджено дворівневу модель кореляції:

- На рівні мікро-кореляції реалізовано алгоритм дедуплікації на основі синтетичних сигнатур, що дозволило зменшити обсяг даних на 40-60% шляхом автоматичного об'єднання ідентичних знахідок від різних сканерів.

- На рівні макро-кореляції (мета-аналізу) впроваджено евристичні правила ("Web Stack", "Critical Cluster"), які автоматично виявляють складні вектори атак шляхом аналізу груп вразливостей.

3. Розроблено та програмно реалізовано адаптивний метод пріоритизації: Створено математичну модель оцінки ризику Priority Score, яка розраховується як зважена сума технічної тяжкості (CVSS), контекстної критичності активу та ступеня достовірності даних. Реалізація динамічного перерахунку ваг через графічний інтерфейс дозволяє адаптувати систему до політики безпеки конкретної організації.

4. Забезпечено інтеграцію з корпоративними системами безпеки: Реалізовано механізм експорту подій у форматі CEF (Common Event Format) через протокол Syslog. Це дозволяє використовувати розроблений інструмент як сенсор для SIEM-систем (наприклад, Elastic Stack, ArcSight), забезпечуючи передачу верифікованих інцидентів до центру моніторингу безпеки (SOC) у режимі реального часу.

5. Створено повнофункціональний інтерфейс користувача: Розроблено ергономічний GUI на базі бібліотеки Tkinter, який надає інструменти для управління активами, запуску сканувань (Nmap, ZAP, Trivy), візуалізації статистики та генерації деталізованих PDF-звітів.

Підсумок: Результатом роботи у третьому розділі є готовий до впровадження програмний продукт, який вирішує ключові проблеми фрагментації даних та інформаційного перевантаження. Система дозволяє перейти від хаотичного накопичення звітів до керованого, ризик-орієнтованого процесу управління вразливостями, інтегрованого в загальну екосистему захисту підприємства.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження проблематики управління вразливостями в сучасних інформаційних системах та розроблено технологію автоматизованої кореляції та пріоритизації результатів сканування.

1. Проведено аналіз теоретичних засад та сучасного стану проблеми. Встановлено, що в умовах стрімкого розвитку інфраструктури (хмарні обчислення, контейнеризація) та зростання інтенсивності кіберзагроз, традиційні підходи до управління вразливостями втрачають ефективність. Визначено, що ключовим бар'єром є не дефіцит інформації про загрози, а неспроможність існуючих засобів ефективно обробляти великі масиви різномірних даних. "Сира" оцінка CVSS, яка є стандартом індустрії, визнана недостатньою для прийняття оперативних рішень, оскільки вона не враховує контекст активу та ступінь достовірності даних.

2. Досліджено інструментальний базис та формати даних. Виконано детальну декомпозицію вихідних даних популярних сканерів з відкритим кодом (Nmap, OWASP ZAP, Trivy). Виявлено фундаментальну проблему семантичної та синтаксичної несумісності звітів: відсутність єдиного ідентифікатора активів (IP vs URL vs Image Name) та використання різних метрик ризику. Доведено, що ізольоване використання цих інструментів призводить до масового дублювання даних, інформаційного перевантаження персоналу («Alert Fatigue») та збільшення часу реакції на інциденти.

3. Розроблено та обґрунтовано нову технологію кореляції. Запропоновано дворівневу модель обробки даних. На рівні мікро-кореляції впроваджено алгоритм генерації синтетичних сигнатур, що забезпечує автоматичну дедуплікацію знахідок. На рівні макро-кореляції (мета-аналізу) розроблено евристичні правила (зокрема "Web Application Stack Correlation"), які дозволяють виявляти складні вектори атак шляхом аналізу перетину множин вразливостей з різних доменів (мережа, додаток, контейнер).

4. Створено адаптивну математичну модель пріоритизації. Розроблено формулу розрахунку інтегрального показника Priority Score, яка базується на зваженій сумі технічної тяжкості вразливості, контекстної критичності активу та коефіцієнта кореляційної достовірності. Ця модель дозволяє динамічно ранжувати інциденти, автоматично знижуючи пріоритет непідтверджених знахідок та фокусуючи увагу на критичних бізнес-активах.

5. Здійснено програмну реалізацію системи. Спроектовано та розроблено автономний програмний комплекс "Vulnerability Correlator" мовою Python. Архітектура системи, побудована на патерні MVC, включає модулі парсингу, базу даних SQLite, аналітичне ядро та графічний інтерфейс на базі Tkinter. Реалізовано функціонал "Smart Import" для автоматичного визначення типів звітів, менеджер шаблонів сканування та генератор професійних PDF-звітів.

6. Забезпечено інтеграцію з екосистемою безпеки підприємства. Реалізовано механізм експорту подій у форматі CEF (Common Event Format) через протокол Syslog, що дозволяє використовувати розроблений інструмент як сенсор для SIEM-систем (Elastic Stack, ArcSight). Експериментально підтверджено успішну передачу та візуалізацію даних у зовнішніх системах моніторингу.

Практичне значення отриманих результатів полягає у створенні готового до впровадження інструменту, який дозволяє автоматизувати рутинні процеси обробки вразливостей, зменшити час виявлення критичних загроз та підвищити загальний рівень захищеності інформаційної системи без необхідності залучення дорогих комерційних рішень. Розроблена технологія може бути ефективно використана у відділах інформаційної безпеки підприємств малого та середнього бізнесу, а також як навчальний посібник для підготовки фахівців з кібербезпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги (ISO/IEC 27001:2013, IDT). Київ : ДП «УкрНДНЦ», 2016. 32 с.
2. Allodi L., Massacci F. Comparing Vulnerability Severity and Exploits Using Case-Control Studies. *ACM Transactions on Information and System Security*. 2014. Vol. 17, No. 1. P. 1–20.
3. Common Vulnerability Scoring System v3.1: Specification Document / FIRST.org. URL: <https://www.first.org/cvss/v3.1/specification-document> (дата звернення: 15.11.2025).
4. Docker Documentation. Containerization platform. URL: <https://docs.docker.com/> (дата звернення: 20.11.2025).
5. Exploit Prediction Scoring System (EPSS) Model Specification. FIRST.org. URL: <https://www.first.org/epss/model> (дата звернення: 18.11.2025).
6. ISO/IEC 27002:2022. Information security, cybersecurity and privacy protection — Information security controls. Geneva : ISO, 2022. 162 p.
7. Lyon G. F. Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Sunnyvale : Insecure.Com LLC, 2009. 468 p.
8. NIST SP 800-40r3. Guide to Enterprise Patch Management Technologies / M. Souppaya, K. Scarfone. Gaithersburg : National Institute of Standards and Technology, 2013. 26 p.
9. Nmap Scripting Engine (NSE) Documentation. URL: <https://nmap.org/book/nse.html> (дата звернення: 10.11.2025).
10. OWASP Top 10 - 2021. The Ten Most Critical Web Application Security Risks. URL: <https://owasp.org/Top10/> (дата звернення: 12.11.2025).

11. OWASP Zed Attack Proxy (ZAP) Documentation. URL: <https://www.zaproxy.org/docs/> (дата звернення: 15.11.2025).
12. Python 3.10 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 01.11.2025).
13. SQLite Documentation. SQL database engine. URL: <https://www.sqlite.org/docs.html> (дата звернення: 05.11.2025).
14. SSVC: Stakeholder-Specific Vulnerability Categorization / J. M. Spring et al. Software Engineering Institute, Carnegie Mellon University. 2020. URL: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=653459> (дата звернення: 25.11.2025).
15. The Penetration Testing Execution Standard (PTES). URL: <http://www.pentest-standard.org/> (дата звернення: 20.11.2025).
16. Trivy Documentation. Comprehensive security scanner. URL: <https://aquasecurity.github.io/trivy/> (дата звернення: 16.11.2025).
17. Tkinter — Python interface to Tcl/Tk. URL: <https://docs.python.org/3/library/tkinter.html> (дата звернення: 02.11.2025).
18. Common Event Format (CEF) Implementation Standard. Micro Focus. URL: <https://www.microfocus.com/documentation/arcsight/arcsight-smartconnectors-8.3/cef-implementation-standard/> (дата звернення: 30.11.2025).
19. Matplotlib: Visualization with Python. URL: <https://matplotlib.org/> (дата звернення: 10.11.2025).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)