Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(освітній рівень)

на тему: "Дослідження методів аналізу шкідливих посилань та IP-адрес у середовищі Linux"

Виконав: студент(ка) 6 курсу, групи СБм - 61

Спеціальності:
125 «Кібербезпека та захист інформації»
(шифр і назва напряму підготовки, спеціальності)
Семчишин Олександр Володимирович
підпис (прізвище та ініціали)

Керівник Козак Р.О.
підпис (прізвище та ініціали)

Нормоконтроль Стадник М.А.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека та захист інформації
(шифр і назва спеціальності)

Студенту Семчишину Олександру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів аналізу шкідливих посилань та IP-адрес у середовищі Linux

Керівник роботи Козак Руслан Орестович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «24» 11 2025 року № 4/7-1024

2. Термін подання студентом завершеної роботи 12.12.2025

3. Вихідні дані до роботи Методи виявлення та аналізу шкідливих посилань та IP-адрес у середовищі Linux

4. Зміст роботи (перелік питань, які потрібно розробити)
Класифікація загроз та каналів витоку даних в середовищі Linux
Аналіз сучасних інструментів та платформ для аналізу індикаторів компрометації (IoC)
Проєктування архітектурної моделі та взаємодії модулів
Реалізація ключових функціональних модулів
Розробка методики експериментальних досліджень
Проведення тестування та аналіз отриманих результатів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність теми, 2. Мета та завдання роботи, 3. Архітектура та потоки даних 4. Реалізація утиліти, 5. Математична модель оцінювання, 6. Тестування утиліти: перевірка посилань, 7. Тестування утиліти: кеш і білий список, 8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., к.т.н., доцент		
Безпека в надзвичайних ситуаціях	Теслюк В.М., проректор з адміністративно–господарської роботи та будівництва		

7. Дата видачі завдання 19.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	19.09 – 22.09	Виконано
2.	Підбір літературних джерел	23.09 – 30.09	Виконано
3.	Опрацювання джерел в галузі дослідження	01.10 – 09.10	Виконано
4.	Аналіз та вибір програмних засобів розробки	10.10 – 16.10	Виконано
5.	Розроблення програмного коду	17.10 – 28.10	Виконано
6.	Тестування роботи утиліти	29.10 – 01.11	Виконано
7.	Оформлення розділу «Аналіз сучасних підходів до виявлення та аналізу шкідливої мережевої активності»	02.11 – 10.11	Виконано
8.	Оформлення розділу «Проєктування архітектури та методів аналізу веб-ресурсів середовищі Linux»	11.11 – 15.11	Виконано
9.	Оформлення розділу «Розробка програмного засобу та дослідження його практичної ефективності»	16.11 – 23.11	Виконано
10.	Виконання завдання до підрозділу «Охорона праці та безпека в надзвичайних ситуаціях»	24.11 – 30.11	Виконано
11.	Оформлення кваліфікаційної роботи	01.12 – 08.12	Виконано
12.	Нормоконтроль	08.12 – 10.12	Виконано
13.	Перевірка на плагіат	12.12 – 14.12	Виконано
14.	Попередній захист кваліфікаційної роботи	17.12.2025	Виконано
15.	Захист кваліфікаційної роботи	23.12.2025	

Студент _____ Семчишин О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Козак Р.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження методів аналізу шкідливих посилань та IP-адрес у середовищі Linux // ОР «Магістр» // Семчишин Олександр Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБм-61 // Тернопіль, 2025 // С. – 65, рис. – 10, додат. – 2, кресл. – 8

Ключові слова: Інформаційна безпека, Моніторинг веб-активності, Аналіз URL-адрес, Виявлення веб-загроз, Фішингові ресурси, Шкідливі веб-сайти, Linux, Rust, Кінцева точка, VirusTotal, URLScan.io, AbuseIPDB, Репутаційний аналіз, Інтегральна оцінка ризику, Кешування результатів.

Кваліфікаційна робота присвячена дослідженню методів моніторингу та оцінювання безпеки веб-ресурсів у середовищі операційних систем сімейства Linux. Актуальність теми зумовлена зростанням кількості фішингових атак та використанням веб-браузерів як основного каналу взаємодії з мережею Інтернет.

У роботі проаналізовано сучасні вектори веб-загроз та підходи до побудови систем моніторингу веб-активності. Розроблено програмний засіб для автоматизованого аналізу відвіданих URL-адрес із використанням зовнішніх сервісів аналізу загроз та інтегральної моделі оцінювання ризику. Проведено експериментальні дослідження для перевірки точності класифікації та оцінки впливу утиліти на продуктивність системи.

Результати роботи можуть бути використані для підвищення рівня безпеки робочих станцій Linux та подальших досліджень у сфері аналізу веб-загроз.

ANNOTATION

Research of Methods for Analyzing Malicious Links and IP Addresses in the Linux Environment // Master's Degree Program // Semchyshyn Oleksandr Volodymyrovych // Ivan Pul'uj Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, group SBm-61 // Ternopil, 2025 // p. 65, figs. 10, apps 2, drws. 8

Keywords: Information security, Web activity monitoring, URL analysis, Web threat detection, Phishing resources, Malicious websites, Linux, Rust, Endpoint security, VirusTotal, URLScan.io, AbuseIPDB, Reputation-based analysis, Integrated risk assessment, Result caching.

The qualification thesis is devoted to the study of methods for monitoring and assessing the security of web resources in operating systems of the Linux family. The relevance of the topic is conditioned by the growing number of phishing attacks and the widespread use of web browsers as the primary channel of interaction with the Internet.

The paper analyzes modern vectors of web-based threats and approaches to the design of web activity monitoring systems. A software tool for automated analysis of visited URLs has been developed, incorporating external threat intelligence services and an integrated risk assessment model. Experimental studies were conducted to verify classification accuracy and to evaluate the impact of the developed utility on system performance.

The results of this work can be applied to enhance the security level of Linux workstations and may serve as a foundation for further research in the field of web threat analysis.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	—	Application Programming Interface,
DLP	—	Data Loss Prevention
DFD	—	Data Flow Diagram
DNS	—	Domain Name System
FP	—	False Positive
FN	—	False Negative
TP	—	True Positive
TN	—	True Negative
TOML	—	Tom's Obvious, Minimal Language

ВСТУП

У сучасному інформаційному просторі, де веб-браузери є основним інструментом доступу до мережі Інтернет, стрімко зростає кількість кіберзагроз, пов’язаних із використанням шкідливих та фішингових веб-ресурсів. Активний розвиток електронних сервісів, дистанційної роботи та хмарних платформ зумовлює підвищені вимоги до безпеки веб-взаємодії користувачів. Операційна система Linux, яка широко застосовується як на робочих станціях, так і у серверній інфраструктурі, дедалі частіше стає об’єктом цілеспрямованих атак через веб-канали. Взаємодія з небезпечними URL-адресами може призвести до компрометації систем, витоку облікових даних, встановлення шкідливого програмного забезпечення та значних фінансових і репутаційних втрат.

Існуючі засоби захисту веб-активності часто орієнтовані на мережевий рівень або використовують окремі репутаційні механізми без урахування контексту дій кінцевого користувача. Такі підходи нерідко характеризуються високим рівнем хибних спрацювань, затримками виявлення загроз або значним навантаженням на системні ресурси. Крім того, значна частина рішень є закритими або недостатньо адаптованими до специфіки Linux-середовища. Це обумовлює необхідність дослідження нових методів моніторингу веб-активності та розробки спеціалізованих інструментів, здатних ефективно аналізувати веб-загрози на рівні кінцевої точки.

У рамках даної роботи здійснюється дослідження архітектурних підходів до побудови систем моніторингу веб-активності, аналізуються сучасні вектори веб-загроз у середовищі Linux та розробляється програмний засіб для автоматизованого оцінювання небезпеки веб-ресурсів. Запропонований підхід базується на агрегуванні результатів зовнішніх сервісів аналізу загроз та використанні інтегральної математичної моделі оцінки ризику, що дозволяє підвищити точність класифікації URL-адрес.

Метою даної кваліфікаційної роботи є підвищення ефективності виявлення та оцінювання небезпеки веб-ресурсів у середовищі Linux шляхом

удосконалення методів моніторингу веб-активності та розробки програмного засобу аналізу URL-адрес.

Для досягнення поставленої мети в роботі передбачається розв’язання таких завдань:

– провести аналітичний огляд сучасних веб-загроз та каналів компрометації через URL-адреси;

– класифікувати типи шкідливих і фішингових веб-ресурсів, актуальних для середовища Linux;

– дослідити можливості використання зовнішніх сервісів аналізу веб-загроз;

– розробити архітектуру та програмну реалізацію утиліти моніторингу веб-активності користувачів;

– запропонувати та реалізувати модель інтегральної оцінки небезпеки веб-ресурсів;

– провести експериментальні дослідження для оцінки точності класифікації, ефективності кешування та впливу утиліти на продуктивність системи.

Предметом дослідження є методи, алгоритми та програмні засоби моніторингу та оцінювання небезпеки веб-ресурсів у середовищі операційних систем сімейства Linux.

Об’єктом дослідження є процес забезпечення безпеки веб-взаємодії користувачів та захисту кінцевих точок від веб-загроз в автоматизованих системах на базі Linux.

Практичне значення отриманих результатів полягає у можливості використання розробленого програмного засобу системними адміністраторами та фахівцями з кібербезпеки для підвищення рівня захисту робочих станцій Linux, а також як основи для подальшої розробки та вдосконалення відкритих систем моніторингу веб-активності.

Публікації. Основні результати кваліфікаційної роботи опубліковано у працях конференції (див. Додаток А).

1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ВИЯВЛЕННЯ ТА АНАЛІЗУ ШКІДЛИВОЇ МЕРЕЖЕВОЇ АКТИВНОСТІ

1.1 Сутність проблеми кіберзагроз, пов'язаних зі шкідливою мережевою активністю

Шкідлива мережева активність становить одну з ключових загроз сучасного інформаційного середовища й характеризується використанням зловмисниками спеціально підготовлених посилань, IP-адрес та мережевої інфраструктури для компрометації систем, викрадення інформації або порушення нормальної роботи сервісів. В операційних системах Linux, які широко функціонують у корпоративних мережах, хмарних платформах та серверних інфраструктурах, такі загрози мають особливо критичний характер, оскільки часто спрямовані на високопродуктивні або публічно доступні ресурси. Для розуміння характеру проблеми необхідно визначити базові поняття, що використовуються у сфері реагування на інциденти та дослідження кіберзагроз [1].

Одним із ключових термінів є індикатор компрометації (Indicator of Compromise, IoC) – З артефакт, який свідчить про можливу шкідливу активність у системі. Такими артефактами можуть бути шкідливі URL, IP-адреси, хеші файлів, специфічні доменні імена чи ознаки підозрілих мережевих з’єднань. Шкідливе посилання визначається як веб-адреса, що веде на ресурс, призначений для фішингу, поширення шкідливого коду або експлуатації вразливостей. Аналогічно, шкідливою IP-адресою вважається адреса, пов’язана з інфраструктурою кіберзлочинців: ботнетами, серверами для DDoS-атак, платформами доставлення експлойтів чи командними серверами. У свою чергу, поняття Threat Intelligence позначає процеси збору, аналізу та інтерпретації даних про загрози з метою підвищення ефективності виявлення та реагування. Важливою частиною такої інфраструктури є C&C-сервери (Command and Control), які забезпечують управління зараженими хостами, передавання команд та ексфільтрацію даних [2, 3].

Шкідливі посилання та IP-адреси відіграють важливу роль у типових сценаріях кібератак, які часто описуються за моделлю Cyber Kill Chain. Модель Cyber Kill Chain, розроблена Lockheed Martin, надає структуровану основу для розуміння етапів, які проходить зловмисник для досягнення своєї мети. Шкідливі IP-адреси та URL відіграють чітко визначені ролі на кожному з цих етапів. Розглянемо типовий сценарій атаки програми-вимагача:

– Фаза 1 – Розвідка.

– Фаза 2 – Озброєння на нібито важливий документ.

– Фаза 3 – Доставка.

– Фаза 4 – Експлуатація.

– Фаза 5 – Встановлення.

– Фаза 6 – Командування та Управління.

– Фаза 7 – Дії по досягненню мети[4].

Аналіз цього ланцюга виявляє асиметричну важливість IoC на різних етапах. IoC на фазі "Доставка" є одноразовим. Якщо його заблокувати після того, як хтось на нього натиснув, це вже запізно. IoC на фазі "Встановлення" також може бути тимчасовим.

Наслідки інцидентів, пов’язаних зі шкідливою мережевою активністю, охоплюють економічні, операційні й репутаційні аспекти. Економічні втрати включають простій систем, обмеження доступу до критичних сервісів, витрати на відновлення та залучення фахівців з кібербезпеки, а також можливі штрафи за витоки даних. Операційні наслідки проявляються у порушенні нормальної роботи інфраструктури, викраденні облікових даних, захопленні серверів для побудови ботнетів або поширення шкідливого трафіку. Репутаційні втрати є не менш значущими, оскільки інциденти негативно впливають на довіру клієнтів, партнерів та інвесторів, що може мати довготривалі фінансові та організаційні наслідки.

Для подолання цих ризиків застосовуються системи виявлення та аналізу загроз, що орієнтовані на моніторинг мережевої активності, виявлення аномалій та автоматизацію реагування. Такі рішення інтегруються з платформами аналізу загроз (Threat Intelligence Platforms, TIP), які забезпечують централізований збір

та кореляцію індикаторів компрометації, інтеграцію зовнішніх джерел даних про загрози та підвищення точності виявлення шкідливих URL та IP-адрес. Використання подібних платформ у середовищі Linux дозволяє систематизувати мережеву інформацію, підвищити якість аналізу інцидентів та забезпечити проактивний захист від кіберзагроз.

1.2 Класифікація загроз та каналів витоку даних в середовищі Linux

Класифікація загроз та векторів атак у середовищі Linux, що пов’язані зі шкідливими посиланнями та IP-адресами, охоплює широкий спектр технік і сценаріїв, які зловмисники використовують для компрометації систем, викрадення даних або залучення серверів до злочинної інфраструктури. Одним із поширених типів загроз є фішинг, що базується на використанні шкідливих URL-адрес для отримання доступу до облікових даних, фінансової інформації або адміністраторських привілеїв. Хоча фішингові атаки здебільшого асоціюються з кінцевими користувачами, серверні Linux-системи також можуть ставати їх об’єктом, наприклад, у разі компрометації веб-панелей керування чи сервісів, до яких адміністратори входять через браузер. Іншим поширеним класом загроз є розповсюдження шкідливого програмного забезпечення, коли шкідливі посилання використовуються для доставлення бекдорів, руткітів або майнерів криптовалют, що здатні працювати у Linux-середовищі. Шкідливі IP-адреси також часто пов’язані з інфраструктурою ботнетів та C&C-серверів, які забезпечують керування зараженими хостами, а у випадках програм-вимагачів формують канали для обміну ключами шифрування або передавання викраденої інформації. Окрему категорію становлять атаки, спрямовані на сканування вразливостей та перебір облікових даних, які здійснюються з відомих шкідливих IP. Такі атаки часто є підготовчим етапом перед подальшими спробами проникнення або розгортанням шкідливих компонентів [3].

Вектори атак у Linux-середовищі демонструють значну різноманітність залежно від цілей зловмисника та характеру інфраструктури. Одним із найпоширеніших напрямів є компрометація веб-серверів, що зумовлена

поширеними конфігураційними помилками, незакритими вразливостями або недостатнім контролем доступу. Зламані сервери часто використовуються для розміщення фішингових сторінок, шкідливих скриптів або редиректів, що спрямовують жертв на інші шкідливі ресурси. Значним є і ризик атак на мережеві сервіси, такі як SSH, FTP або сервери баз даних, які зазвичай доступні в публічних мережах і формують основний вектор brute-force атак із використанням шкідливих IP-адрес. Після успішної компрометації такі сервери можуть перетворюватися на частину ботнет-мережі та використовуватися для подальших деструктивних дій – проведення DDoS-атак, масового розсилання спаму, розповсюдження шкідливих посилань серед інших учасників мережі. У сучасних інфраструктурах, що активно використовують контейнеризацію та хмарні обчислення, дедалі актуальнішими стають загрози, пов’язані з шкідливою активністю всередині контейнерів або хмарних інстансів. Такі середовища часто динамічні, мають автоматизовані механізми масштабування та складні внутрішні мережі, що створює додаткові можливості для прихованого розгортання шкідливих компонентів або встановлення несанкціонованих з’єднань із шкідливими IP-адресами [1].

Специфіка Linux як серверної платформи обумовлює важливість глибокого аналізу системних журналів для виявлення та класифікації загроз. Логи syslog та auth.log дозволяють ідентифікувати спроби несанкціонованих входів, аномальні мережеві підключення, підозрілі запуски процесів та ознаки брутфорс-активності. Журнали веб-серверів, такі як журнали доступу та помилок Apache або Nginx, містять детальну інформацію про запити, що можуть вказувати на спроби сканування вразливостей, ін’єкційні атаки або виклики шкідливих URL. Паралельно з аналізом журналів важливим є моніторинг мережевого трафіку, який дозволяє фіксувати звернення до шкідливих IP, встановлення підозрілих TCP/UDP-сесій або інші аномалії, характерні для C&C-комунікацій. Значну роль відіграє й конфігурація брандмауера, зокрема iptables і nftables, які забезпечують можливість блокування небажаного трафіку, фільтрації пакетів, створення правил для обмеження доступу та виявлення потенційно небезпечних мережевих взаємодій. У поєднанні ці механізми формують основу ефективного виявлення

та класифікації загроз, що використовують шкідливі посилання та IP-адреси, забезпечуючи проактивний підхід до захисту Linux-систем.

1.3 Аналіз сучасних інструментів та платформ для аналізу індикаторів компрометації (IoC)

Ефективна протидія витокам даних вимагає впровадження спеціалізованих програмних комплексів, здатних здійснювати глибоку інспекцію контенту та контекстний аналіз операцій з інформацією. Сучасні DLP-системи будуються за модульним принципом, де кожен компонент відповідає за захист даних у певному стані. За точкою застосування контролю DLP-рішення класифікують на три основні категорії: Network DLP, Storage DLP та Endpoint DLP [2].

Аналіз сучасних інструментів та платформ для дослідження індикаторів компрометації (IoC) є ключовим елементом забезпечення кібербезпеки у Linux-середовищі, оскільки саме від ефективності роботи цих рішень залежить своєчасне виявлення загроз, встановлення їхнього походження та запобігання подальшій шкоді. Індикатори компрометації, такі як шкідливі URL-адреси, IP, доменні імена, хеші файлів чи артефакти мережевого трафіку, стають основою для автоматизації виявлення та реагування. Сучасна екосистема інструментів для аналізу IoC охоплює кілька категорій рішень, кожна з яких виконує специфічні завдання та доповнює інші технології у загальній системі захисту.

Однією з найважливіших категорій є зовнішні сервіси репутації, які забезпечують можливість перевірки підозрілих посилань або IP-адрес на основі великомасштабних баз знань. До таких сервісів належать VirusTotal, AbuseIPDB, PhishTank, Spamhaus та інші ресурси, що агрегують дані з різних антивірусних рушіїв, систем виявлення загроз і дослідницьких платформ. Linux-адміністратори та фахівці з аналізу інцидентів активно застосовують ці сервіси як початковий етап перевірки IoC, оскільки вони дозволяють отримати інформацію про відомі випадки зловживання адресою, її участь у ботнетах, фішингових кампаніях або розповсюдженні шкідливого ПЗ. Результати таких

перевірок можуть бути використані для блокування трафіку, автоматичного оновлення фільтрів або поповнення внутрішніх систем моніторингу [5, 6].

– VirusTotal.

– AbuseIPDB.

– PhishTank.

– Spamhaus.

Ці сервіси не є взаємозамінними, оскільки вони мають різні моделі даних та рівні довіри. Розуміння цих відмінностей є критичним для правильної інтеграції. Іншу важливу групу становлять платформи аналізу загроз (Threat Intelligence Platforms), такі як MISP або TheHive у поєднанні з Cortex. Ці рішення забезпечують централізоване збирання, систематизацію та кореляцію індикаторів компрометації з кількох джерел, включно з OSINT-ресурсами, комерційними фідами або внутрішніми джерелами організації. Завдяки підтримці структурованих форматів обміну, наприклад, STIX/TAXII, такі платформи спрощують інтеграцію із системами моніторингу, SIEM-рішеннями та інструментами автоматизованого реагування. У Linux-середовищі вони набувають особливого значення, оскільки дають змогу впроваджувати централізовані політики щодо блокування шкідливих адрес, аналізувати тенденції атак та отримувати актуальні звіти про активність зловмисників [9].

Важливою складовою інфраструктури безпеки є системи аналізу мережевого трафіку, зокрема мережеві системи виявлення та попередження атак (NIDS/NIPS), такі як Suricata та Zeek. Ці інструменти працюють у режимах реального часу та забезпечують можливість виявлення підозрілих з’єднань, аналізу структури запитів, виявлення звернень до шкідливих IP або C&C-серверів, а також фіксації аномалій у поведінці мережевих служб Linux-систем. Suricata використовує сигнатури та правила для швидкого виявлення відомих загроз, тоді як Zeek орієнтований на глибокий поведінковий аналіз трафіку. Разом вони забезпечують широкі можливості для відстеження IoC та подальшої інтеграції результатів аналізу з платформами Threat Intelligence [12].

Окремою групою інструментів є так звані “пісочниці”, що застосовуються для динамічного аналізу URL-адрес, шкідливих сайтів та потенційно

небезпечних файлів, які можуть бути завантажені з таких посилань. Пісочниці забезпечують ізольоване середовище, у якому можна безпечно виконати шкідливий код або дослідити поведінку веб-ресурсу, зокрема перенаправлення, спроби експлуатації вразливостей чи звернення до C&C-інфраструктури. Результати такого аналізу дозволяють отримати деталізоване уявлення про загрозу, визначити її технічні характеристики та сформувати нові індикатори компрометації для подальшого моніторингу у Linux-системах [10].

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МЕТОДІВ АНАЛІЗУ ВЕБ-РЕСУРСІВ У СЕРЕДОВИЩІ LINUX

2.1 Обґрунтування вибору технологічного інструментарію та архітектурних рішень

Створення інструменту для автоматизованого моніторингу веб-активності користувача, виявлення потенційно шкідливих ресурсів та формування структурованих подій безпеки передбачало побудову комплексу компонентів різного функціонального призначення. Вибір технологічного стеку для такого програмного забезпечення є ключовим, оскільки саме обрані мови та бібліотеки визначають швидкодію, безпечність, масштабованість та подальшу придатність системи до розширення. З огляду на те, що утиліта працює з історією браузера Firefox, взаємодіє з локальною SQLite-базою, здійснює багатопотокові зовнішні запити до платформ кіберзагроз і в реальному часі формує події, вимоги до технологій були нетривіальними та комплексними [11].

В основі архітектури лежить ідея чіткого розмежування двох рівнів програмної системи: невидимого для користувача низькорівневого модуля аналізу та високорівневого інтерфейсу, орієнтованого на зручність і швидкість взаємодії. У процесі порівняльного аналізу різних мов програмування було встановлено, що жодна з них не має одночасно і максимальної швидкості обробки, і простоти побудови веб-інтерфейсів. Саме тому архітектурне рішення полягало у комбінації Rust і Python – двох мов, які належать до різних парадигм, але доповнюють одна одну.

Rust був обраний у якості основи системи не через популярність, а через набір властивостей, які є критично важливими саме для фонових моніторингових сервісів. На відміну від традиційних мов системного рівня, таких як C або C++, Rust гарантує пам’яткову безпеку на етапі компіляції. Його borrow-checker забезпечує відсутність гонок даних, некоректного доступу до пам’яті, висячих покажчиків і помилок управління життєвим циклом об’єктів. Це означає, що навіть при багатопотоковій роботі – а утиліта активно використовує

паралельність – система не схильна до типових для багатопотокових програм багів [16].

У процесі аналізу історії браузера Rust демонструє продуктивність, близьку до C, але при цьому позбавлену його недоліків у вигляді складності налагодження та ймовірності критичних помилок. Оскільки Firefox веде історію у SQLite-базі «places.sqlite», читання таких файлів у реальному часі потребує мінімальних пауз та гарантій незмінності даних. Rust забезпечує оптимальний доступ до бази, що дозволяє уникнути блокувань і виконувати SQL-запити з максимальною ефективністю, не створюючи навантаження на браузер [14].

В окрему перевагу Rust варто віднести його мінімальний розмір виконуваних файлів, відсутність залежності від віртуальних машин або сторонніх рантаймів, та здатність працювати в автономному режимі. Це критично важливо для аналітичної утиліти, що може використовуватись у середовищах із обмеженим доступом до інтернету або з жорсткими вимогами до безпеки [15].

На відміну від системного модуля, інтерфейс sysadmin- або analyst-рівня має абсолютно інші вимоги. Він повинен бути гнучким, швидким у розробці, адаптивним до змін структури даних та зручним для користувача. Python надає унікальні можливості для роботи з форматом JSONL, який обрано для зберігання подій безпеки. Він дозволяє легко парсити, фільтрувати, сортувати та візуалізувати дані, що надходять від аналітичного модуля [20].

Flask ж, як мінімалістичний і легкий фреймворк, дозволяє побудувати веб-інтерфейс без зайвих залежностей, не накладаючи жорстких обмежень на структуру проєкту. На відміну від «монолітних» рішень, таких як Django, Flask не вимагає складної конфігурації, не створює надлишкових абстракцій і дозволяє розробнику повністю контролювати архітектуру маршрутизації.

Важливим аспектом є також ізоляція Python-модуля від Rust-ядра: аналітика може оновлюватися незалежно від інтерфейсу й навпаки, що позитивно впливає на гнучкість у майбутньому. Ця модульність забезпечує можливість масштабування проекту у напрямку SOAR-платформи з подальшим додаванням

користувацьких правил реагування, автоматичних playbook-сценаріїв, генерації звітів тощо.

2.2 Проєктування архітектурної моделі та взаємодії модулів

Загальна логіка роботи утиліти ґрунтується на подійній моделі. Кожен новий запис у базі історії переглядів розглядається як окрема подія, що ініціює повний цикл аналізу. На відміну від класичних антивірусних рішень або мережевих фільтрів, утиліта не втручається у трафік у момент передачі, а працює з наслідком взаємодії користувача з веб-середовищем. Такий підхід дозволяє уникнути складної інтеграції з мережевими драйверами, проксі-серверами або браузерними розширеннями та, водночас, забезпечує достатній рівень інформативності для оцінювання ризиків.

Архітектура утиліти побудована за модульним принципом, який передбачає наявність окремих слабко зв’язаних компонентів, кожен з яких виконує чітко визначену функцію. У загальному вигляді систему можна подати як сукупність трьох логічних рівнів: рівня збору даних, рівня аналізу та рівня представлення результатів користувачеві. Такий поділ дозволяє ізолювати складні обчислювальні процеси від інтерфейсу, а також спрощує тестування й супровід програмного продукту.

Аналітичне ядро системи реалізовано як окремий фоновий процес операційної системи. Воно запускається незалежно від браузера та веб-інтерфейсу й виконує безперервний цикл перевірки нових записів у базі історії. Такий підхід дозволяє досягти високої автономності: навіть якщо веб-інтерфейс тимчасово не запущений, процес аналізу продовжує працювати та накопичувати події у журналі.

Доступ до файлу places.sqlite здійснюється у режимі тільки для читання. Це принципово важливо з точки зору безпеки та стабільності роботи браузера, оскільки утиліта не має змоги змінити або пошкодити історію переглядів. Для автоматичного визначення розташування профілю користувача використовується механізм пошуку системних директорій, що дозволяє утиліті

коректно працювати незалежно від версії операційної системи та конкретного користувача [15].

Внутрішня логіка аналітичного ядра побудована таким чином, щоб уникати повторної обробки одних і тих самих записів. Для цього після кожного успішного циклу аналізу фіксується часовий маркер останнього обробленого переходу. Під час наступного запуску або ітерації система звертається до бази даних із відповідним обмеженням, отримуючи лише ті записи, що з'явилися після попереднього проходу. Це забезпечує високу продуктивність навіть при великому обсязі історії переглядів [16].

Кожен новий веб-ресурс, отриманий із бази, спочатку проходить етап попередньої обробки. На цьому етапі виконується приведення URL-адреси до уніфікованого вигляду, видалення надлишкових параметрів і виділення ключових структурних компонентів. Такий підхід дозволяє зменшити кількість дубльованих записів і підвищити точність подальшого аналізу.

Після нормалізації запускається етап репутаційної перевірки. Для цього система звертається до зовнішніх сервісів аналізу загроз, які надають інформацію про наявність ресурсу у базах фішингових, шкідливих або компрометованих сайтів. Отримані результати не використовуються безпосередньо у «чистому» вигляді, а слугують вхідними даними для формування узагальненої оцінки ризику.

Оцінка ризику формується на основі сукупності факторів, серед яких кількість негативних детекцій, тип виявлених загроз, частота появи ресурсу в історії переглядів та контекст відвідування. Завдяки цьому система не реагує однаково на всі підозрілі ресурси, а дозволяє виділяти дійсно критичні події.

Після завершення аналізу формується структурована подія безпеки, яка містить інформацію про URL-адресу, час відвідування, результати перевірок і підсумковий рівень ризику. Ця подія записується у журнал у вигляді окремого незалежного запису. Формат збереження забезпечує можливість подальшого автоматизованого аналізу, фільтрації та візуалізації результатів.

Після того як подія сформована та зафіксована у журналі, вона стає доступною для наступного рівня архітектури – рівня представлення та взаємодії

з користувачем. Цей рівень реалізовано у вигляді веб-інтерфейсу, який працює як самостійний програмний модуль. Принциповим архітектурним рішенням є те, що веб-інтерфейс не має прямого зв’язку з аналітичним ядром. Уся взаємодія між ними здійснюється виключно через файл журналу подій. Такий підхід дозволяє зберегти незалежність компонентів, значно підвищує стійкість системи до збоїв і спрощує її супровід.

Веб-інтерфейс функціонує як локальний веб-застосунок, доступ до якого здійснюється через стандартний браузер користувача. Після запуску сервер ініціалізує власний процес прослуховування локального мережевого порту та переходить у режим очікування запитів. Користувач відкриває інтерфейс як звичайну веб-сторінку, після чого отримує доступ до переліку зафіксованих подій безпеки. Така організація дозволяє використовувати утиліту без встановлення додаткового спеціалізованого клієнтського програмного забезпечення [16].

Логіка відображення інформації у веб-інтерфейсі побудована таким чином, щоб користувач отримував не просто набір “сирих” записів, а структуровану картину поточної ситуації. Події групуються за часовими мітками, рівнем ризику та доменами. Це дозволяє швидко оцінювати загальний стан безпеки, виявляти аномалії та концентрувати увагу на найбільш критичних інцидентах. Окремі події можуть бути відкриті для детального перегляду, де відображається повна інформація про ресурс, результати перевірок, отримані з зовнішніх сервісів, а також підсумкове рішення системи щодо рівня небезпеки.

Оновлення інформації у веб-інтерфейсі реалізовано у режимі, близькому до реального часу. Через визначені інтервали браузер надсилає запит до сервера для отримання оновленого списку подій. Сервер, у свою чергу, повторно зчитує журнал подій, аналізує зміни та повертає лише актуальну інформацію. Завдяки цьому користувач бачить нові інциденти майже одразу після їх фіксації аналітичним ядром, не виконуючи ручного оновлення сторінки.

З погляду архітектури важливо, що веб-інтерфейс не змінює первинні дані. Журнал подій розглядається як незмінне джерело істини. Будь-які дії користувача, такі як підтвердження інциденту, його переведення у статус

опрацьованого або перенесення до архіву, не впливають на первинний запис. Для цього використовується окреме службове сховище станів, у якому зберігаються лише зміни статусів. Такий підхід відповідає принципам збереження цілісності доказової інформації, що є важливим у контексті інформаційної безпеки та можливого проведення розслідувань.

Користувацька взаємодія з інтерфейсом побудована таким чином, щоб мінімізувати когнітивне навантаження. Основні дії, які доступні оператору, можна узагальнити так:

– перегляд списку інцидентів у хронологічному порядку;

– фільтрація подій за рівнем ризику;

– перегляд детальної інформації про окремий ресурс;

– зміна статусу події (наприклад, “новий”, “опрацьований”, “хибне спрацювання”);

– перенесення застарілих подій до архіву.

Ці дії не змінюють сам механізм роботи аналітичного ядра, проте дозволяють користувачу формувати власну модель контролю над ситуацією.

Окремої уваги в архітектурі приділено питанням безпеки доступу до веб-інтерфейсу. Оскільки сервер працює локально, базовим механізмом захисту є обмеження доступу лише з локального хоста. Усі сесії користувача мають обмежений життєвий цикл, а передані між браузером і сервером дані проходять процедури перевірки та екранування, що унеможливлює виконання шкідливих скриптів навіть у випадку, якщо у журналі містяться спеціально сформовані підозрілі значення.

Питання цілісності та надійності збереження даних займає центральне місце в архітектурі утиліти. Формат JSON Lines було обрано не випадково. Він поєднує у собі одразу декілька важливих властивостей: простоту, читабельність, потоковість та стійкість до пошкоджень. Кожен запис існує незалежно від інших, тому пошкодження одного рядка не призводить до втрати усього журналу. Крім того, такий формат зручний для подальшої обробки сторонніми інструментами, що відкриває можливості інтеграції утиліти з іншими системами безпеки.

Запис подій у журнал здійснюється за принципом “тільки додавання”. Жоден уже створений запис не може бути змінений або видалений аналітичним ядром. Це забезпечує незмінність історії спостережень і дозволяє відновлювати повну хронологію дій навіть через тривалий проміжок часу. Особливо важливою ця властивість є у випадку, коли виникає потреба проаналізувати розвиток інциденту заднім числом.

Архітектура утиліти також передбачає механізми відмовостійкості. Якщо у процесі роботи аналітичного ядра відбувається збій, після повторного запуску система автоматично повертається до останнього збереженого стану та продовжує обробку подій з того місця, на якому була перервана. Аналогічно, у випадку перезапуску веб-інтерфейсу усі події знову стають доступними після повторного зчитування журналу. Таким чином жоден інцидент не втрачається навіть за умов аварійних ситуацій.

Потік даних можна представити у вигляді наступної схеми, див. рис. 2.1.

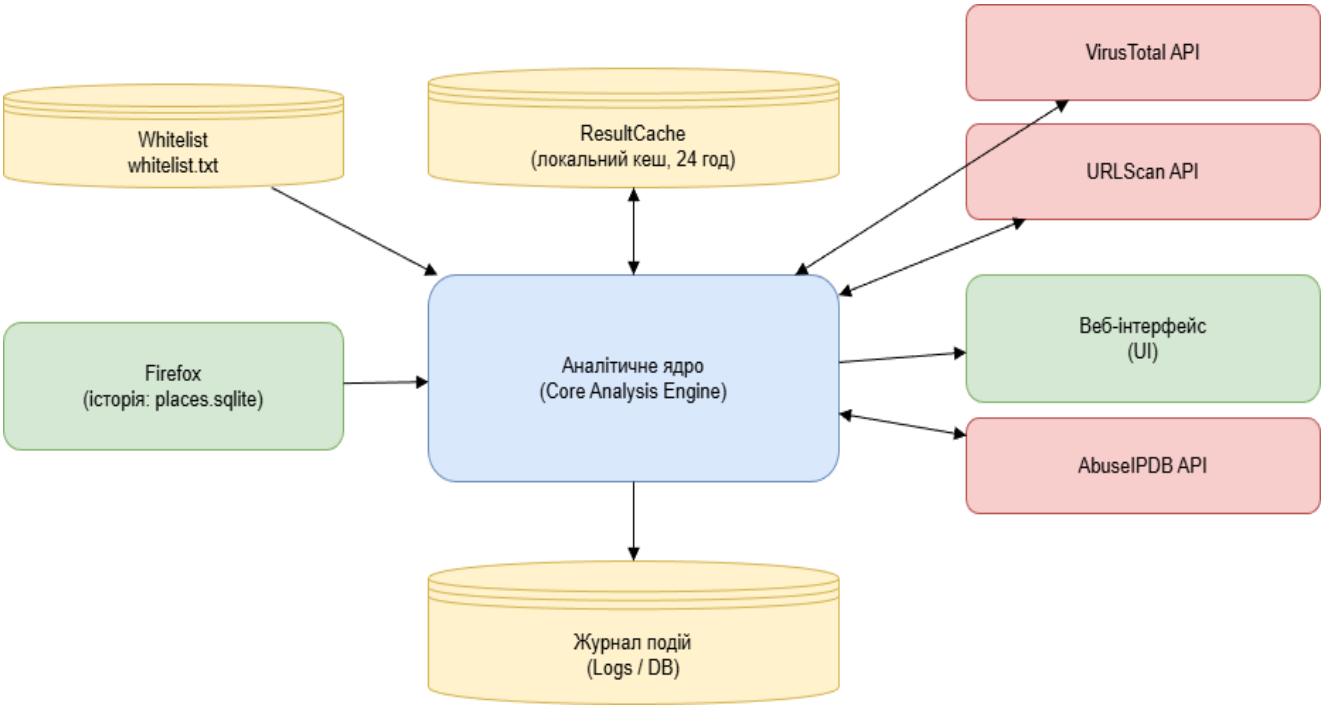

Рисунок 2.1 – Діаграма потоку даних утиліти моніторингу веб-ресурсів

Важливою складовою архітектури є механізм роботи з зовнішніми сервісами аналізу загроз. Система спроєктована таким чином, щоб не залежати критично від їхньої доступності. Якщо у певний момент часу звернення до

сервісу не вдалося, подія не відкидається, а помічається як така, що потребує повторної перевірки. При наступних циклах аналізу система може повторити запит і доповнити інформацію новими результатами. Такий підхід дозволяє зберігати працездатність утиліти навіть в умовах нестабільного мережевого з'єднання.

Окремим аспектом архітектурного проєктування є можливість масштабування. Хоча утиліта орієнтована на локальне використання, її архітектура не обмежується виключно цим сценарієм. За потреби журнал подій може бути винесений на віддалений сервер, а веб-інтерфейс – розгорнутий як централізована панель моніторингу для кількох робочих станцій. У такому випадку змінюється лише спосіб доступу до журналу, тоді як внутрішня логіка аналітичного ядра залишається без змін. Це підкреслює гнучкість і адаптивність обраної архітектурної моделі.

Загалом реалізована архітектура поєднує у собі риси класичних систем збору подій безпеки та сучасних локальних систем поведінкового аналізу. Вона не перевантажена зайвими компонентами, не потребує складної інфраструктури, але водночас забезпечує достатній рівень функціональності для практичного використання. Саме таке поєднання – мінімальна складність при достатній аналітичній глибині – і визначає доцільність обраних архітектурних рішень.

Особливе місце в загальній архітектурі утиліти займають питання безпеки зберігання даних та захисту конфігураційної інформації. Оскільки система працює з результатами аналізу потенційно шкідливих веб-ресурсів, будь-яка компрометація журналів або ключів доступу до зовнішніх сервісів може призвести не лише до втрати функціональності, а й до створення нових загроз. Саме тому в архітектурі закладено принцип розмежування доступу: конфігураційні файли із ключами доступу зберігаються окремо від журналів подій, а права доступу до них обмежуються на рівні файлової системи. Аналітичне ядро зчитує ці параметри лише під час запуску, після чого вони існують виключно в оперативній пам'яті та не потрапляють у файли журналу.

Передача даних до зовнішніх сервісів виконується лише через захищені мережеві з'єднання. Це дозволяє унеможливити перехоплення запитів або

підміну відповідей у процесі обміну даними. Незважаючи на те, що веб-інтерфейс працює локально, у його логіці також передбачено базові механізми захисту від несанкціонованого доступу, зокрема обмеження доступу лише для локального користувача та контроль сесій. Таким чином система не створює додаткових векторів атаки у межах операційної системи.

Важливою особливістю архітектури є захист журналу подій від несанкціонованих змін. Журнал формується за принципом незмінного сховища: події лише додаються, але ніколи не змінюються та не перезаписуються. Це дозволяє розглядати його не просто як лог-файл, а як своєрідний хронологічний реєстр подій веб-активності. Навіть якщо користувач у веб-інтерфейсі змінює статус інциденту або переносить його в архів, первинний запис залишається незмінним. Такий підхід відповідає вимогам до зберігання доказової інформації в системах інформаційної безпеки.

Архітектура утиліти також враховує можливі збої як програмного, так і апаратного характеру. Якщо під час запису події відбувається аварійне завершення процесу, втрачається лише поточний запис, що перебував у буфері, тоді як уся попередня історія зберігає цілісність. При наступному запуску аналітичне ядро автоматично відновлює роботу з останнього зафіксованого часового маркера. Таким чином навіть за умов нестабільного живлення або некоректного завершення роботи операційної системи утиліта не втрачає критичних даних.

Аналогічні принципи закладено і у роботі веб-інтерфейсу. Якщо під час перегляду або обробки інцидентів сервер перезапускається, після повторного запуску користувач знову отримує доступ до всіх подій, що містяться у журналі. Жодна інформація не зберігається виключно у пам’яті без резервного дублювання у файлах. Така модель суттєво підвищує надійність у повсякденній експлуатації.

Окрему увагу в архітектурі приділено поведінці системи у випадку нестабільної роботи зовнішніх сервісів аналізу загроз. Оскільки доступ до них залежить від мережевого з’єднання та обмежень на кількість запитів, утиліта не розглядає їх як єдине джерело істини. Якщо відповідь від сервісу не була

отримана або надійшла з помилкою, подія не вважається остаточно опрацьованою. Вона отримує проміжний статус та може бути перевірена повторно під час наступних циклів аналізу. Це дозволяє уникнути ситуацій, коли через тимчасову недоступність сервісу подія безпідставно класифікується як безпечна.

З точки зору довготривалої експлуатації важливу роль відіграє механізм архівації. У міру накопичення подій активний журнал може зростати до значних обсягів, що ускладнює його оперативну обробку. Тому архітектура передбачає перенесення застарілих або вже опрацьованих інцидентів до архівного сховища. Архів розглядається не як втрачена інформація, а як довготривале сховище для проведення ретроспективного аналізу, аудиту або навчальних досліджень. Такий підхід дозволяє підтримувати баланс між продуктивністю та повнотою історичних даних.

Масштабованість системи врахована ще на етапі її архітектурного проєктування. Хоча базовий варіант утиліти орієнтований на локальне використання на одному комп'ютері, її логіка не прив'язана жорстко до цього сценарію. За потреби аналітичне ядро може працювати як окремий агент, який передає журнал подій на віддалений сервер, де вже розгортається централізований веб-інтерфейс. У такому випадку змінюється лише транспорт доставки журналу, тоді як внутрішня логіка обробки подій залишається незмінною. Це відкриває перспективи використання утиліти не лише у персональних, а й у корпоративних середовищах.

Важливою характеристикою архітектури є її відносна універсальність. Усі ключові етапи обробки – зчитування джерела, нормалізація, репутаційна перевірка, оцінка ризику, журналювання та візуалізація – реалізовані як логічно відокремлені ланки. Це дозволяє за потреби замінити одне джерело даних іншим без повної перебудови всієї системи. Наприклад, замість Firefox може бути використаний інший браузер або окремий файл логів мережевої активності, а замість одного сервісу Threat Intelligence – інший. Архітектура при цьому зберігає свою цілісність.

У підсумку реалізована архітектура утиліти є збалансованою з точки зору складності та функціональних можливостей. Вона не потребує складної серверної інфраструктури, не використовує сторонні хмарні платформи та не залежить від специфічного апаратного забезпечення. Водночас вона забезпечує повний цикл моніторингу веб-активності – від фіксації факту відвідування ресурсу до прийняття рішення про рівень його небезпеки та подальшого аналізу інциденту користувачем.

Таким чином, запропонована архітектура дозволяє розглядати утиліту не просто як допоміжну програму, а як цілісну локальну систему підтримки прийняття рішень у сфері інформаційної безпеки. Вона може бути використана як у навчальних цілях, так і як основа для створення більш складних рішень класу SOC або SOAR у майбутньому. Обрані архітектурні підходи демонструють, що навіть у рамках відносно простої програмної реалізації можливо досягти поєднання надійності, гнучкості та практичної корисності.

2.3 Схематичне представлення функціональних процесів утиліти

Отже, як уже було описано, розроблена утиліта моніторингу веб-активності користувача побудована на основі модульної архітектури, яка складається з п’яти основних компонентів. Модуль конфігурації відповідає за завантаження та валідацію параметрів системи з конфігураційного файлу, забезпечуючи гнучке налаштування всіх аспектів роботи програми. Модуль збору даних реалізує зчитування історії веб-переглядів із локальної бази даних браузера Firefox places.sqlite та відповідає за виявлення нових записів про відвідані ресурси. Модуль аналізу здійснює нормалізацію URL-адрес, формування запитів до зовнішніх сервісів репутаційної перевірки та визначення рівня небезпеки веб-ресурсів. Модуль типів даних визначає структури подій безпеки та рівні ризику, забезпечуючи типобезпечну роботу всієї системи. Головний модуль координує роботу всіх компонентів системи та керує життєвим циклом програми. Схему роботи утиліти представлено на рисунку 2.2.

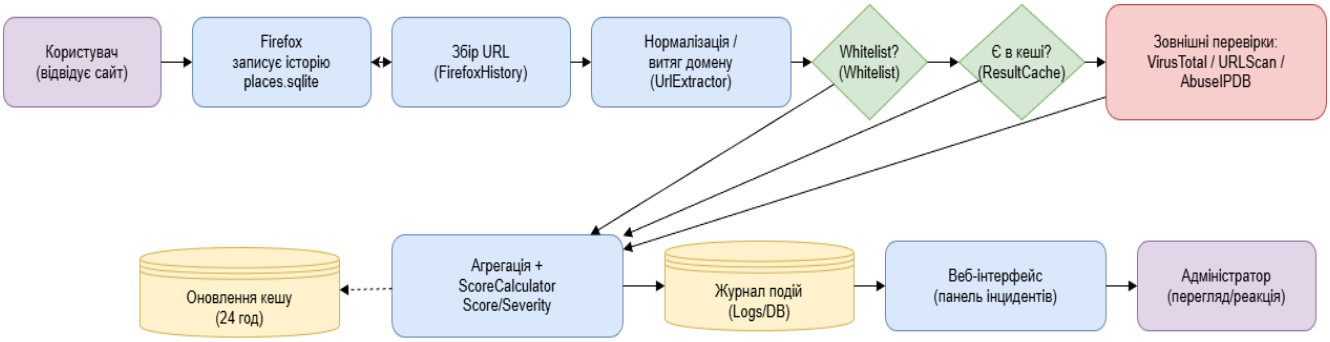

Рисунок 2.2 – Схема роботи утиліти моніторингу веб-активності

Процес обробки веб-переходів у системі можна представити у вигляді скінченного автомату з послідовними станами. Початковим станом є відвідування користувачем веб-ресурсу, що фіксується браузером Firefox та зберігається у локальній базі історії переглядів. Система моніторингу виявляє новий запис у базі даних та передає інформацію про веб-ресурс до наступного етапу обробки. Здійснюється перевірка коректності URL-адреси, під час якої система аналізує її формат та вміст для визначення можливості подальшого аналізу. Якщо адреса не відповідає встановленим критеріям, вона ігнорується системою, і процес обробки завершується.

Якщо веб-ресурс пройшов первинну перевірку, виконується його нормалізація та підготовка до репутаційної перевірки. Далі система здійснює аналіз ресурсу шляхом звернення до зовнішніх сервісів аналізу загроз, застосовуючи всі налаштовані механізми перевірки для виявлення ознак фішингу, шкідливого програмного забезпечення або компрометованої інфраструктури. За результатами перевірки можливі два сценарії розвитку подій. У випадку відсутності негативних детекцій веб-ресурс отримує статус безпечного, інформація про це записується у журнал подій, і обробка завершується успішно. Якщо ж виявлено ознаки небезпеки, система переходить до аналізу їх критичності.

Аналіз критичності полягає у визначенні найвищого рівня ризику серед отриманих результатів перевірки. Для подій з рівнем ризику Medium або Low система обмежується логуванням інформації про потенційно підозрілий ресурс, дозволяючи користувачу продовжувати роботу без обмежень. Якщо ж виявлено рівень ризику Critical або High, подія фіксується як інцидент безпеки з

підвищеною загрозою та відображається у веб-інтерфейсі з відповідною позначкою для привернення уваги користувача.

Запуск утиліти відбувається у чітко визначеній послідовності етапів, кожен з яких є критичним для коректної роботи системи. Перший етап полягає у завантаженні конфігурації з конфігураційного файлу, під час якого система зчитує та парсить усі параметри роботи. До цих параметрів належать шляхи збереження журналів подій, інтервал перевірки історії браузера, параметри доступу до зовнішніх сервісів аналізу загроз та правила формування рівнів ризику. У разі помилки на цьому етапі робота програми припиняється, оскільки подальше функціонування без коректної конфігурації є неможливим.

Другий етап передбачає ініціалізацію модуля аналізу, під час якого здійснюється підготовка структур для роботи з URL-адресами, налаштування механізмів з’єднання із зовнішніми сервісами та підготовка внутрішніх структур для обробки результатів перевірок. Ініціалізація виконується лише один раз під час запуску, що суттєво підвищує продуктивність подальшої роботи системи.

На третьому етапі створюється агент моніторингу веб-активності, який ініціалізує структури даних для відстеження вже оброблених записів історії переглядів та запускає лічильники статистики для збору показників роботи. Також на цьому етапі перевіряється доступність файлу бази places.sqlite та готовність системи до переходу в активний режим роботи.

Четвертий етап полягає у початковому аналізі вже наявної історії веб-переглядів. Система зчитує всі записи за визначений початковий період та виконує їх повну перевірку, що дозволяє виявити потенційно небезпечні ресурси ще до початку активного моніторингу нових переходів.

П’ятий завершальний етап активує систему моніторингу веб-активності у реальному часі. Аналітичне ядро переходить у режим періодичного опитування бази даних історії переглядів, а веб-інтерфейс запускається для відображення інцидентів користувачу. Одночасно активується механізм збору статистики, який формує звіти про роботу системи через задані проміжки часу. Після завершення всіх етапів ініціалізації система переходить у режим активного моніторингу та готова до безперервної обробки подій веб-активності.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ТА ДОСЛІДЖЕННЯ ЙОГО ПРАКТИЧНОЇ ЕФЕКТИВНОСТІ

3.1 Опис середовища розробки та тестування

Ефективність роботи розробленої утиліти безпосередньо залежить від характеристик обчислювального середовища, у якому вона функціонує, оскільки програмний модуль постійно взаємодіє з системними ресурсами, підсистемами введення-виведення, планувальником процесів та файловою системою. Для забезпечення стабільності експериментів, відтворюваності результатів та усунення впливу сторонніх чинників було сформовано окреме стандартизоване середовище розгортання та тестування.

Як платформа для виконання утиліти використовувалося віртуалізоване середовище на базі гіпервізора VMware. Такий підхід дозволив ізолювати програму від основної операційної системи, забезпечити керованість обчислювальних ресурсів та створити можливість швидкого повернення до контрольного стану за допомогою механізмів знімків віртуальної машини. Це є важливим для проведення серійних експериментів та аналізу поведінки програми за різних умов навантаження.

Гостьова операційна система розгорнута на базі дистрибутива Ubuntu Linux версії 24.04 LTS. Використання актуальної версії операційної системи обумовлене наявністю сучасного ядра Linux, що забезпечує покращену роботу підсистеми введення-виведення, оптимізовані механізми керування пам’яттю та актуальні засоби захисту. Крім того, дана версія має повну сумісність з сучасними інструментами розробки та бібліотеками мови програмування Rust.

Віртуальній машині було виділено два віртуальних процесорних ядра (2 vCPU), що емулюють архітектуру x86_64. Така конфігурація є достатньою для виконання аналітичних операцій, обробки запитів до зовнішніх сервісів та роботи веб-інтерфейсу без суттєвих затримок. Обсяг оперативної пам’яті становить 8 ГБ, що дозволяє зберігати значні обсяги проміжних даних у пам’яті,

виконувати паралельні обчислення та забезпечувати стабільну роботу системи навіть за підвищеного навантаження [25].

Дискова підсистема віртуальної машини реалізована через віртуальний накопичувач формату VMDK, який фізично розміщений на твердотільному накопичувачі хост-системи. Таке рішення дозволяє мінімізувати затримки операцій введення-виведення, що є критично важливим під час роботи з журналами подій, файлами конфігурації та базами даних браузера [25].

Процес розробки виконувався з використанням стабільної гілки компілятора Rust. Для керування залежностями, компіляції та запуску тестових сценаріїв застосовувалася стандартна система збірки Cargo. Це дозволило забезпечити контроль версій бібліотек, відтворюваність результатів компіляції та автоматизацію процесу перевірки працездатності програмного коду.

Оскільки стандартні налаштування операційної системи орієнтовані на звичайний режим роботи користувача, додатково була виконана оптимізація параметрів ядра, що впливають на обробку системних подій та спостереження за ресурсами. Зокрема, були збільшені ліміти для підсистеми файлових сповіщень з метою забезпечення стабільної роботи утиліти в режимі постійного моніторингу. Це дозволило уникнути втрати подій та забезпечити коректну фіксацію змін у системі протягом тривалого часу без зниження продуктивності.

3.2 Реалізація ключових функціональних модулів

Модуль firefox_history.rs призначений для отримання списку нещодавно відвіданих веб-ресурсів користувача з локальної бази даних браузера Firefox. Він виконує роль первинного джерела даних для всієї системи, оскільки саме на основі отриманих URL-адрес у подальшому здійснюється репутаційний аналіз та формування інцидентів безпеки.

В основі модуля лежить структура FirefoxHistory, яка інкапсулює шлях до активного профілю Firefox. Під час ініціалізації виконується автоматичний пошук профілю користувача з урахуванням різних варіантів встановлення

браузера (Snap, Flatpak та класична інсталяція). Це дозволяє уникнути ручного налаштування та забезпечує універсальність роботи утиліти.

Доступ до бази історії places.sqlite реалізовано через створення тимчасової копії файлу. Такий підхід виключає можливі конфлікти з браузером, який може блокувати оригінальну базу під час роботи. Подальша обробка виконується за допомогою бібліотеки rusqlite, що забезпечує коректне підключення до SQLite-бази та виконання SQL-запитів.

Витягнення самих URL-адрес реалізовано у методі query_urls, де відбувається відкриття SQLite-бази та виконання SQL-запиту до таблиці moz_places. Запит сформовано таким чином, щоб одразу відсікти значну частину технічного шуму [16]

Лістинг 3.1 – Запит до бази даних браузера

```
SELECT DISTINCT url FROM moz_places
WHERE url LIKE 'http%'
  AND url NOT LIKE '%google-analytics%'
  AND url NOT LIKE '%doubleclick%'
  AND url NOT LIKE '%facebook.com/tr%'
  AND url NOT LIKE '%analytics%'
  AND last_visit_date IS NOT NULL
ORDER BY last_visit_date DESC
LIMIT ...
```

За результатами роботи модуль повертає впорядкований і очищений список URL-адрес, які відображають реальну веб-активність користувача. Завдяки цьому FirefoxHistory забезпечує надійний та безпечний механізм збору початкових даних для всієї системи моніторингу.

Модуль кешування, реалізований у файлі result_cache.rs, призначений для тимчасового збереження результатів перевірки веб-ресурсів з метою зменшення кількості повторних запитів до зовнішніх сервісів аналізу та підвищення загальної продуктивності утиліти. Він дозволяє повторно використовувати результати перевірок URL-адрес, якщо вони залишаються актуальними протягом визначеного проміжку часу [15].

Основною структурою модуля є CachedResult, яка зберігає детальну інформацію про результати аналізу веб-ресурсу, зокрема кількість шкідливих,

підозрілих, невиявлених та безпечних детекцій, часову мітку перевірки, підсумкову оцінку ризику, рівень критичності та додаткові аналітичні коефіцієнти:

Лістинг 3.2 – Структура кешу

```
pub struct CachedResult {
    pub malicious: u32,
    pub suspicious: u32,
    pub undetected: u32,
    pub harmless: u32,
    pub checked_at: DateTime<Utc>,
    pub score: f64,
    pub severity: String,
    pub v_prog: f64,
    pub uncertainty: f64,
    pub analysis: f64,
}
```

Для зберігання множини таких результатів використовується структура ResultCache, яка містить асоціативний масив, де ключем є URL-адреса, а значенням – відповідний результат аналізу. Кеш автоматично зберігається у файлі .firefox_monitor_cache.json, що розміщується в домашньому каталозі користувача.

Під час ініціалізації модуль виконує завантаження кешу з файлу та автоматично відфільтровує застарілі записи, старші за 24 години. Це забезпечує використання лише актуальних результатів і запобігає накопиченню неактуальних даних. Якщо формат кешу не відповідає поточній версії структури, він автоматично очищується.

Метод get() дозволяє отримати кешований результат для конкретного URL за умови, що він ще не втратив актуальність. Метод insert() використовується для додавання нового результату до кешу разом з автоматичним збереженням у файл. Додатково передбачено метод clear_old(), який видаляє всі застарілі записи.

Таким чином, модуль кешування забезпечує оптимізацію роботи утиліти шляхом зменшення кількості звернень до зовнішніх сервісів, прискорює повторний аналіз одних і тих самих ресурсів та підвищує стабільність роботи

системи в умовах обмежень на кількість запитів або нестабільного мережевого з'єднання.

Модуль config.rs призначений для централізованого зчитування та зберігання параметрів конфігурації утиліти, необхідних для її коректної роботи. Він забезпечує гнучке налаштування програми без необхідності зміни програмного коду, шляхом використання змінних середовища операційної системи.

До обов'язкових параметрів належить ключ доступу до сервісу VirusTotal (VT_API_KEY), без якого робота утиліти є неможливою. Ключі для сервісів AbuseIPDB та Urlscan є необов'язковими та зчитуються як опціональні значення. Інтервал сканування (SCAN_INTERVAL) визначає періодичність перевірки історії браузера, а параметр PROCESS_NAME дозволяє вказати назву процесу браузера, який контролюється системою [16].

Зчитування конфігурації реалізовано у методі from_env(), який отримує значення безпосередньо зі змінних середовища:

Лістинг 3.3 – Підключення API VirusTotal

```
let api_key = env::var("VT_API_KEY")
    .map_err(|_| ConfigError::MissingApiKey)?;
```

Якщо обов'язковий ключ VT_API_KEY відсутній, система генерує помилку та припиняє подальшу роботу. Для інших параметрів передбачені значення за замовчуванням, що дозволяє запускати утиліту навіть за мінімальної конфігурації.

У типовому випадку фрагмент відповіді VirusTotal для URL має вигляд, який подібний до лістингу 3.4.

Лістинг 3.4 – Фрагмент відповіді VirusTotal

```
{
  "data": {
    "id": "aHR0cHM6Ly9leGFtcGxlLmNvbQ",
    "type": "url",
    "attributes": {
      "last_analysis_stats": {
        "harmless": 62,
```

Продовження лістингу 3.4

```
          "malicious": 5,
          "suspicious": 2,
          "undetected": 9,
          "timeout": 0
        },
        "last_analysis_date": 1715604321,
        "reputation": -15,
        "categories": {
          "Sophos": "malware",
          "BitDefender": "phishing"
        }
      }
    }
}
```

В свою чергу утиліта використовує лише параметри з кількістю фіксацій:

Лістинг 3.5 – Потрібні параметри

```
"last_analysis_stats": {
  "harmless": 62,
  "malicious": 5,
  "suspicious": 2,
  "undetected": 9
}
```

При відправленні URL на аналіз модуль дотримується вимог API, використовуючи форму application/x-www-form-urlencoded замість JSON. Для отримання звіту за вже відомою адресою використовується стандартна схема кодування URL у форматі Base64 без заповнення (URL-safe, без padding), що реалізовано таким чином, як зазначено у лістингу 3.6.

Лістинг 3.6 – Запит за допомогою Base64

```
    use base64::{Engine as _,
engine::general_purpose::URL_SAFE_NO_PAD};

    // Кодування URL у формат Base64 (URL-safe, без padding)
    let url_id = URL_SAFE_NO_PAD.encode(url.as_bytes());

    // Формування запиту для отримання звіту
    let check_url = format!(
        "https://www.virustotal.com/api/v3/urls/{}",
        url_id
    );

    let response = client
```

Продовження лістингу 3.6

```
        .get(&check_url)
        .header("x-apikey", &self.api_key)
        .send()?;
```

Модуль передбачає обробку типових помилок сервісу: невалідний API-ключ, перевищення лімітів запитів, відсутність запису в базі або інші помилки HTTP-рівня. У разі таких ситуацій користувач отримує зрозумілі діагностичні повідомлення.

У підсумку, даний модуль забезпечує прозору інтеграцію утиліти з сервісом VirusTotal: він перетворює зовнішню JSON-відповідь на компактну структуру VTStats, яка вже безпосередньо використовується алгоритмами оцінювання ризику для класифікації веб-ресурсів за рівнем небезпеки.

Модуль білого списку, реалізований у файлі whitelist.rs, призначений для зберігання та перевірки довірених доменів, які не підлягають подальшому репутаційному аналізу. Його основна мета полягає у зменшенні кількості хибних спрацювань системи та оптимізації процесу перевірки за рахунок виключення завідомо безпечних ресурсів. Вигляд біого списку представлено на рисунку 3.1.

```
# Whitelist довірених доменів
# Формат: один домен на рядок
# Коментарі починаються з #
# Підтримуються субдомени (напр. google.com включає maps.google.com)

google.com
youtube.com
github.com
stackoverflow.com
wikipedia.org
mozilla.org
reddit.com
twitter.com
facebook.com
linkedin.com
dou.ua
```

Рисунок 3.1 – Вигляд файлу білого списку

В основі модуля використовується структура Whitelist, яка містить множину доменів типу HashSet<String>. Використання структури HashSet забезпечує швидку перевірку належності домену до білого списку з амортизованою складністю доступу, як показано у лістингу 3.7.

Лістинг 3.7 – Реалізація білого списку

```
    if domain == *whitelisted || domain.ends_with(&format!(".{}", 
whitelisted)) {
        return true;
```

Модуль url_extractor.rs призначений для автоматичного виділення веб-адрес із довільного тексту та приведення їх до уніфікованого формату. Його основним завданням є пошук потенційних URL-адрес у результатах аналізу, логах або текстових повідомленнях з подальшою підготовкою цих адрес до репутаційної перевірки.

В основі модуля використовується структура UrlExtractor, яка інкапсулює скомпільований регулярний вираз для пошуку доменних імен.

Під час ініціалізації модуля в конструкторі new() створюється регулярний вираз, який дозволяє знаходити як повні URL із зазначенням протоколу, так і адреси без нього. Патерн орієнтований на типову структуру доменних імен та підтримує багаторівневі домени.

Метод extract_urls(&self, text: &str) перебирає всі знайдені збіги в переданому тексті та формує список URL-адрес. Якщо знайдена адреса не містить протоколу, до неї автоматично додається префікс https://, що забезпечує уніфікований формат для подальшої обробки.

Лістинг 3.8 – Нормалізація посилання

```
if !url.starts_with("http") {
    url = format!("https://{}", url);
}
```

Окремо реалізовано метод normalize_url, який дозволяє привести будь-який рядок до стандартного вигляду URL шляхом додавання протоколу за потреби.

Це спрощує інтеграцію модуля з іншими компонентами системи та запобігає помилкам під час репутаційного аналізу.

Модуль інтеграції з сервісом URLScan.io реалізує додатковий рівень аналізу веб-ресурсів, доповнюючи дані VirusTotal поведінковою інформацією про сторінку. На відміну від простого репутаційного сервісу, URLScan виконує повноцінне “проглядання” сторінки у ізольованому середовищі, формуючи детальний звіт про її структуру, мережеві взаємодії та потенційні ознаки шкідливої активності.

Перевірка URL реалізується методом check_url(), що виконує повний цикл взаємодії із сервісом: відправлення URL на сканування, очікування завершення аналізу та отримання підсумкового звіту. Після запуску сканування клієнт витримує паузу у 10 секунд, а потім до трьох разів пробує отримати результати, роблячи проміжні затримки у випадку, якщо звіт ще не сформовано. У разі невдачі користувачу повертається інформативне повідомлення про те, що результати не вдалося отримати.

Для відправлення запиту на сканування використовується структура ScanRequest, яка серіалізується у JSON та містить URL та режим видимості («public»). Відповідь на запит сканування описується структурою ScanResponse, у якій зберігається унікальний ідентифікатор сканування (UUID) та службова API-адреса.

Отриманий згодом детальний звіт десеріалізується у структуру ScanResult, яка містить вкладений блок verdicts та інформацію про сторінку (page). Усередині verdicts визначено кілька рівнів деталізації: загальний вердикт (overall), окремий блок urlscan, а також результати по окремих аналізаторах у engines. Для зручності подальшої обробки всі ці дані агрегуються у спрощену структуру URLScanResult, яка містить тільки ті поля, що безпосередньо використовуються системою: підсумковий бал, ознаку шкідливості, кількість шкідливих та “безпечних” рушіїв, список категорій, домен, IP-адресу та країну.

Особливістю цього модуля є необхідність обробки нестандартних полів відповіді API. У деяких випадках поля на кшталт categories, brands або tags можуть бути представлені як масив рядків, а в інших – як булеве значення false

або null. Для коректної десеріалізації таких даних використовується спеціальна функція deserialize_string_vec_or_bool, що перетворює масив рядків у Vec<String>, а всі інші варіанти (false, null, неочікувані типи) – у порожній вектор. Це дозволяє спростити подальшу логіку обробки без необхідності додаткових перевірок у бізнес-логіці.

У типовому випадку відповідь URLScan.io на запит результатів має вкладену структуру, яку можна умовно подати у вигляді, як показано у лістингу 3.9.

Лістинг 3.9 – Шаблон відповіді URLScan

```
{
  "uuid": "0190f0b6-7c7b-7e6a-bf3a-9b0e3b1f9e52",

  "verdicts": {
    "overall": {
      "score": -45,
      "malicious": true,
      "categories": [
        "phishing",
        "malware"
      ],
      "brands": [
        "PayPal"
      ],
      "tags": [
        "login-page",
        "credential-harvesting"
      ]
    },

    "urlscan": {
      "score": -30,
      "malicious": true,
      "categories": [
        "phishing"
      ],
      "brands": false,
      "tags": [
        "forms"
      ]
    },

    "engines": {
      "score": -60,
      "malicious": [
        "PhishTank",
        "OpenPhish"
```

Продовження лістингу 3.9

```
      ],
      "malicious_total": 2,
      "benign_total": 18
    }
  },

  "page": {
    "domain": "example-phish.com",
    "ip": "185.203.116.42",
    "country": "RU",
    "server": "nginx"
  }
}
```

Саме ці дані перетворюються на структуру URLScanResult, яка вже безпосередньо використовується для формування інтегральної оцінки ризику та збагачення інцидентів контекстною інформацією (домен, IP, країна, категорії).

Додатково для зручності відлагодження та виведення результатів реалізовано трейт Display для URLScanResult, що дозволяє в компактній формі показати основні показники сканування (бал, ознаку шкідливості, домен).

Модуль інтеграції з сервісом AbuseIPDB реалізує перевірку IP-адрес на наявність зловмисної активності, що дозволяє доповнювати аналіз URL-адрес інформацією про репутацію відповідних хостів. На відміну від сервісів, що працюють безпосередньо з URL, AbuseIPDB оперує саме IP-адресами, агрегуючи звіти користувачів та провайдерів безпеки про зловживання.

Для представлення відповіді API використовується структура AbuseIPDBResponse з вкладеними полями AbuseIPDBData, які відображають основні характеристики IP-адреси: рівень довіри (abuse_confidence_score), кількість звітів, країну, тип використання, провайдера, домен та інші атрибути. Типовий фрагмент відповіді сервісу має вигляд, як зазначено у лістингу 3.10.

Лістинг 3.10 – Шаблон відповіді AbuseIPDB

```
{
  "data": {
    "ipAddress": "185.203.116.42",
    "isPublic": true,
    "ipVersion": 4,
    "isWhitelisted": false,
    "abuseConfidenceScore": 78,
```

Продовження лістингу 3.10

```
    "countryCode": "UA",
    "usageType": "Data Center/Web Hosting/Transit",
    "isp": "Example Hosting Ltd",
    "domain": "example-hosting.com",
    "hostnames": [
      "example-phish.com",
      "cdn.example-phish.com"
    ],
    "totalReports": 124,
    "numDistinctUsers": 47,
    "lastReportedAt": "2025-02-18T09:41:22+00:00"
  }
}
```

Ці дані десеріалізуються у внутрішні структури, після чого агрегуються у спрощену форму AbuseIPDBResult, яка використовується іншими модулями системи.

Клієнт AbuseIPDBClient інкапсулює API-ключ та HTTP-клієнт reqwest. Основний метод check_ip() перевіряє коректність IP-адреси, формує запит до API з параметром maxAgeInDays=90 (аналіз останніх 90 днів), обробляє можливі помилки (невалідний ключ, перевищення лімітів, некоректний запит) та перетворює JSON-відповідь на структуру AbuseIPDBResult. Додатково вводиться просте правило інтерпретації результатів: IP-адреса вважається відносно безпечною, якщо вона внесена до білого списку сервісу або її abuse_confidence_score менший за 25.

Окремо реалізовано допоміжну логіку для роботи з URL-адресами. Метод extract_ip_from_url() спочатку розбирає URL, виділяє ім’я хоста, а далі, у разі потреби, виконує DNS-резолюцію до IP-адреси. На основі цього методу функція check_url() дозволяє прозоро перевіряти не лише IP-адреси, але й повні URL, автоматично витягуючи з них відповідний IP.

Для зручності відлагодження та журналювання реалізовано трейт Display для AbuseIPDBResult, що дозволяє у компактному вигляді відображати IP-адресу, відсотковий рівень підозрілості (abuse score) та кількість звітів.

3.3 Розробка методики експериментальних досліджень

Для об’єктивної оцінки небезпеки веб-ресурсів у розробленій утиліті використовується математична модель інтегрального показника безпеки, реалізована у модулі оцінювання ScoreCalculator. На відміну від простого порогового підходу (наприклад, «якщо знайдено хоча б один детект – вважати ресурс шкідливим»), дана модель враховує одразу два ключові фактори: частку виявлених загроз та якість проведеного аналізу. Це дозволяє отримати узагальнену числову оцінку Score у діапазоні від 0 до 10, яка надалі інтерпретується як рівень загрози.

Базовою структурою, що описує результат оцінювання, є SecurityScore, яка містить фінальний бал, рівень загрози та допоміжні показники (прогресивну оцінку виявлень, фактор аналізу тощо.

Інтерпретація значення score виконується через дискретизацію на три рівні загрози за допомогою переліку Severity:

– Low (0–4) – ресурс вважається безпечним;

– Medium (4–8) – ресурс класифікується як підозрілий;

– High (8–10) – ресурс розглядається як небезпечний.

Функція Severity::from_score() визначає рівень ризику на основі числового значення інтегрального показника, що дозволяє уніфікувати трактування результатів для подальшого відображення у веб-інтерфейсі.

Математична модель розрахунку інтегрального показника ґрунтується на агрегованих даних зовнішніх сервісів аналізу (VirusTotal, URLScan, AbuseIPDB). Вхідними параметрами є:

– detected (k) – кількість позитивних детекцій (шкідливих або підозрілих спрацювань);

– total (N) – загальна кількість рушіїв або перевірок;

– analysis_quality (An) – коефіцієнт якості проведеного аналізу в діапазоні [0;1], що відображає глибину та повноту перевірки.

На першому етапі розраховується базове співвідношення виявлених загроз:

$$V\_raw = \frac{k}{N} \tag{3.1}$$

Цей показник відображає частку рушіїв, які класифікували ресурс як шкідливий або підозрілий. Однак його пряме використання є недоцільним, оскільки при високих значеннях k невеликі зміни можуть суттєво впливати на інтерпретацію результату. Тому для згладжування та нелінійного посилення впливу високих значень використовується логістична функція, що формує прогресивну оцінку виявлень

$$V\_prog = \frac{1}{1+e^{-s(V_{raw}-m)}} \tag{3.2}$$

де s – параметр крутизни логістичної кривої, а m – точка перегину (середина). У реалізації обрано значення s=10. та m= 0.5, що забезпечує плавне зростання при помірних значеннях V_raw та різке посилення оцінки при досягненні частки детекцій понад 50%. Таким чином, поодинокі спрацювання не призводять до різкого збільшення інтегрального балу, але при концентрації детекцій понад половину рушіїв система реагує значно агресивніше.

Другим компонентом моделі є фактор аналізу An (analysis_quality), який відображає узагальнену оцінку якості проведеного аналізу. Він може враховувати такі аспекти, як кількість задіяних рушіїв, наявність додаткових поведінкових даних, результати кількох незалежних сервісів тощо. У реалізації модулю передбачається, що це значення вже нормалізоване до інтервалу [0;1].
Фінальна інтегральна оцінка Score у оновленій версії моделі обчислюється як зважена лінійна комбінація двох вказаних компонентів із подальшою нормалізацією до шкали [0;10]:

$$Score = 10 \cdot (0.75 \cdot V\_prog + 0.25 \cdot An) \tag{3.3}$$

У даній формулі коефіцієнт 0.75 надає найбільшу вагу прогресивній оцінці виявлень, що відображає домінуючу роль фактичних детекцій у прийнятті рішення. Фактор аналізу з вагою 0.25 дозволяє додатково коригувати результат залежно від контексту та глибини перевірки. Після розрахунку значення Score обмежується інтервалом [0;10], що гарантує коректну інтерпретацію незалежно від крайніх випадків.

Для спрощених сценаріїв передбачено окремий метод calculate_simple(), який дозволяє оцінити ризик лише на основі співвідношення detected/total. У цьому випадку параметр analysis_quality визначається евристично залежно від частки детекцій: при високих значеннях співвідношення задається більша вага, при низьких – менша. Такий режим може застосовуватися для попереднього або прискореного аналізу, коли детальної інформації ще недостатньо.

Запропонована оновлена модель інтегральної оцінки ризику дозволяє перейти від жорстких бінарних рішень «безпечний/небезпечний» до більш гнучкої, контекстно залежної шкали, яка краще відображає реальний стан загрози. Це, у свою чергу, дає змогу ранжувати інциденти, виділяти найкритичніші випадки для першочергової обробки та підвищувати якість прийняття рішень у системі моніторингу веб-активності.

3.4 Проведення тестування та аналіз отриманих результатів

Для перевірки результативності створеної утиліти аналізу веб-ресурсів було розроблено узагальнену схему експериментального тестування, яка передбачає оцінювання як якісних характеристик (точність визначення безпечних і потенційно небезпечних URL-адрес), так і кількісних параметрів (його вплив на швидкодію системи). Запропонований підхід ґрунтується на репутаційному аналізі ресурсів із використанням узагальнених даних зовнішніх сервісів та подальшим обчисленням інтегрованого показника рівня безпеки.

Оскільки утиліта виконує класифікацію веб-ресурсів на основі трирівневої шкали (Low – Medium – High), для формальної верифікації результатів

використовується матриця помилок (Confusion Matrix), адаптована до задачі аналізу URL:

– True Positive (TP) – шкідливий або фішинговий URL коректно ідентифікований як небезпечний;

– False Positive (FP) – безпечний URL помилково класифікований як небезпечний;

– True Negative (TN) – безпечний URL коректно визначений як безпечний;

– False Negative (FN) – шкідливий URL помилково віднесений до безпечних.

Тестування варто розпочати з Набору «безпечних» URL. Метою даного тесту є перевірка здатності системи коректно ідентифікувати легітимні веб-ресурси та мінімізувати хибні спрацювання (False Positive). Для цього Формується контрольний набір URL-адрес відомих безпечних ресурсів: офіційні сайти програмного забезпечення, освітні платформи, веб-ресурси відкритих проєктів, державні портали. Кеш результатів попередньо очищений.

Очікуваним результатом буде отримати для кожного URL:

– низьке значення Score у межах 0–4;

– рівень загрози Severity = Low;

– відсутність інцидентів високого або середнього рівня.

В нашому випадку усі перевірені ресурси були коректно класифіковані як безпечні. Хибних спрацювань не зафіксовано, що свідчить про високий рівень специфічності системи., див. рис. 3.1.

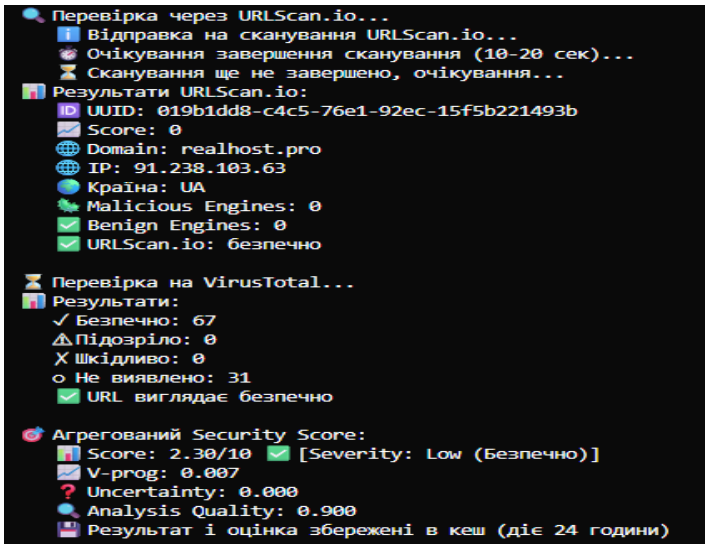

Рисунок 3.1 – Результат сканування легітимних посилань

Метою даного тесту є перевірка здатності системи виявляти небезпечні веб-ресурси та коректно відносити їх до високого рівня загрози. Для цього формується контрольний набір URL-адрес, що містять активні фішингові сторінки, сайти з розповсюдженням шкідливого програмного забезпечення та ресурси з високим рівнем репутаційного ризику згідно з даними VirusTotal, URLScan та AbuseIPDB. Кеш результатів попередньо очищений.

Очікуваним результатом є отримання для кожного URL:

– високого значення Score у межах 8–10;

– рівня загрози Severity = High;

– фіксації інциденту небезпечного рівня.

У ході тестування всі перевірені шкідливі ресурси були коректно ідентифіковані системою та віднесені до категорії небезпечних. Пропусків загроз (False Negative) зафіксовано не було, що свідчить про високу чутливість системи до реальних загроз, див. рис. 3.2, 3.3.

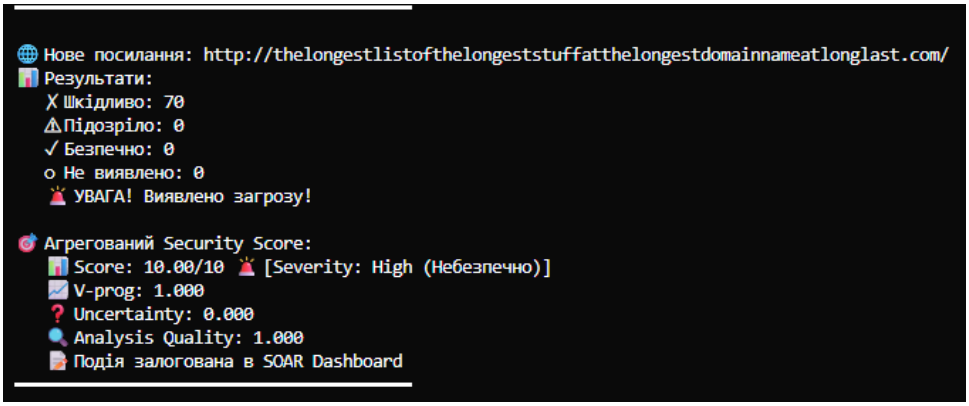

Рисунок 3.2 – Результат сканування шкідливого посилання

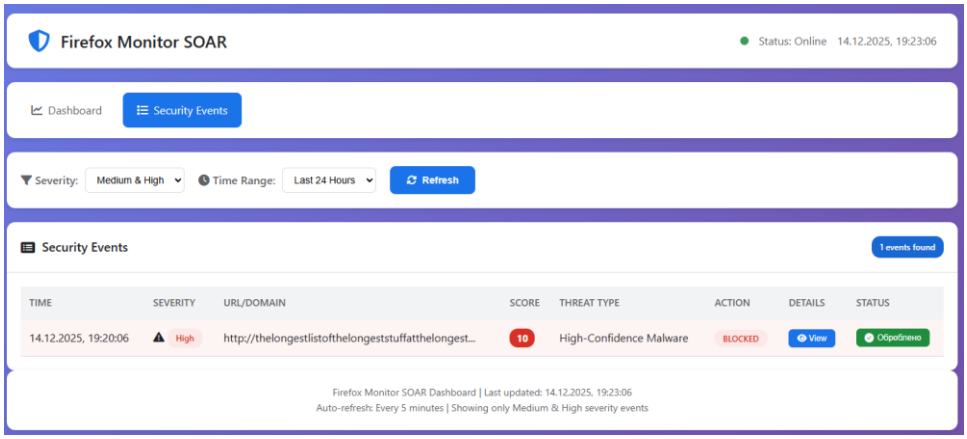

Рисунок 3.3 – Було створено подію безпеки в системі логування

Метою даного тесту є перевірка коректності роботи механізму білого списку доменів та його впливу на процес аналізу URL-адрес. Для цього частина доменів із безпечного та потенційно підозрілого наборів була додана до файлу whitelist.txt. Після цього відповідні веб-ресурси повторно відкривалися у браузері Firefox, а кеш результатів залишався очищеним.

Очікуваним результатом є:

– автоматичне ігнорування URL, домени яких знаходяться у whitelist;

– відсутність звернень до зовнішніх сервісів аналізу;

– присвоєння статусу безпечного ресурсу незалежно від попередніх результатів.

У результаті тестування всі URL-адреси з білого списку були коректно пропущені без детального аналізу та класифіковані як безпечні. Некоректних спрацювань механізму whitelist зафіксовано не було, як зображено на рисунку 3.4.

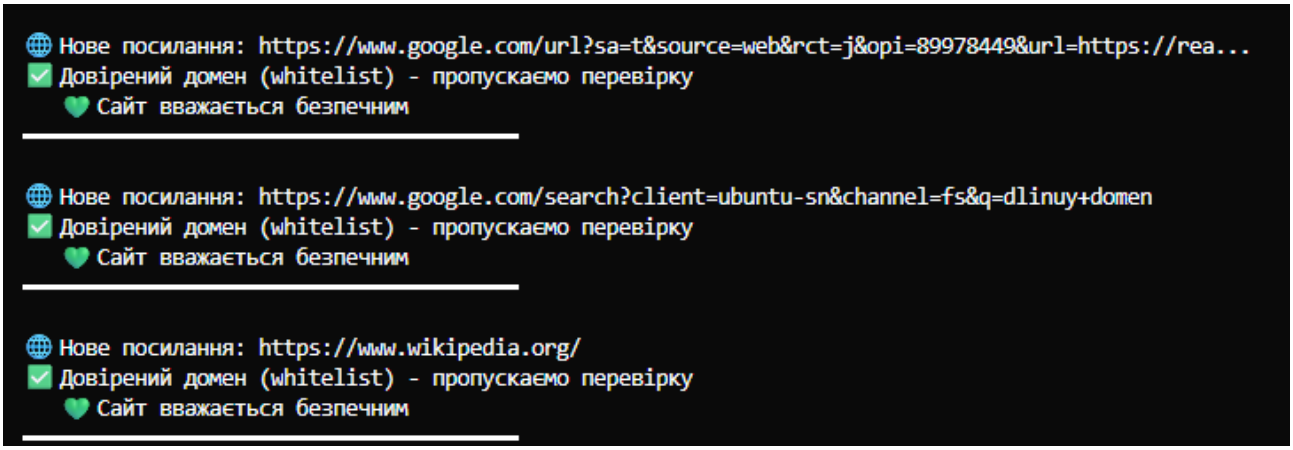

Рисунок 3.4 – Посилання з білого списку

Метою даного тесту є перевірка ефективності механізму кешування результатів перевірок та зменшення кількості повторних звернень до зовнішніх сервісів. Для цього один і той самий набір URL-адрес перевірявся двічі з невеликим часовим інтервалом. Перед першим запуском кеш був повністю очищений.

Очікуваним результатом є:

– виконання повноцінної перевірки через зовнішні API при першому запуску;

– відсутність повторних HTTP-запитів до VirusTotal, URLScan та AbuseIPDB при повторній перевірці;

– зменшення часу обробки URL за рахунок використання локального кешу.

У ході тестування було встановлено, що при повторній перевірці утиліта використовувала лише локально збережені результати без виконання нових запитів до API. Це дозволило суттєво скоротити час аналізу та уникнути перевищення лімітів сервісів., див. рис. 3.5.

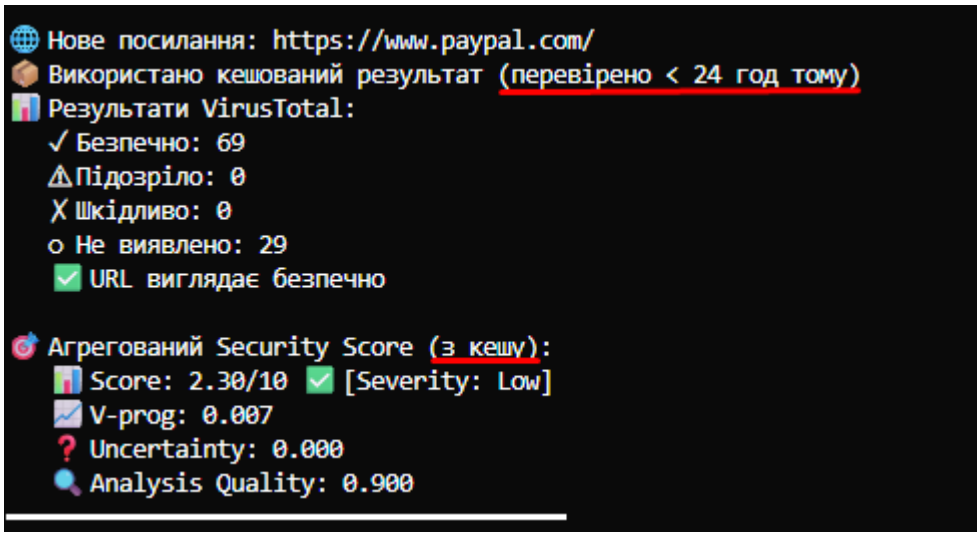

Рисунок 3.5 – Посилання витягнуті з кешу

Навантажувальне тестування – метою даного тесту є оцінка стабільності роботи утиліти та її впливу на продуктивність системи за умов інтенсивної веб-активності. Для цього у браузері Firefox за короткий проміжок часу відкривалася велика кількість різних веб-ресурсів із безпечних та підозрілих наборів, при цьому утиліта працювала у режимі безперервного моніторингу.

Очікуваним результатом є:

– стабільна робота утиліти без зависань та аварійних завершень;

– відсутність критичного навантаження на процесор і оперативну пам’ять;

– коректна обробка всіх нових записів історії браузера без втрати подій.

У результаті експерименту встановлено, що утиліта працює стабільно навіть при високій інтенсивності веб-переходів, не створює критичного навантаження на систему та не призводить до пропуску подій моніторингу, див. рис. 3.6.

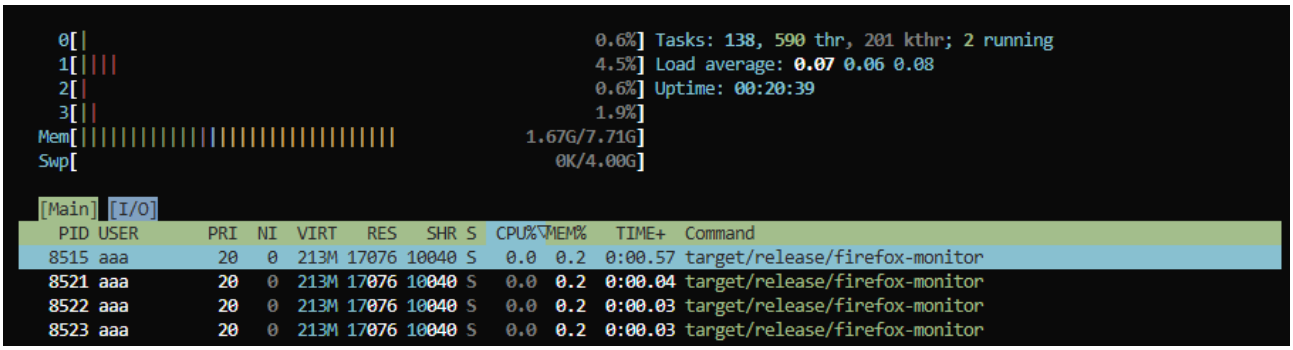

Рисунок 3.6 – Використання ресурсів ПК утилітою

У результаті проведеного тестування було підтверджено працездатність розробленої утиліти та її здатність виконувати репутаційний, статичний і динамічний аналіз веб-ресурсів та IP-адрес у середовищі Linux. Експериментальні випробування показали, що система коректно обробляє вхідні дані, застосовує інтеграцію з зовнішніми платформами та внутрішніми правилами, а також формує узагальнений показник ризику, що дозволяє оцінювати потенційно шкідливі адреси з достатнім рівнем точності.

Таким чином, узагальнення отриманих даних підтверджує доцільність практичної реалізації утиліти та обґрунтовує її застосування для аналітичних і дослідницьких завдань у сфері виявлення шкідливої мережевої активності. Запропонована система демонструє потенціал для подальшого вдосконалення, зокрема шляхом розширення джерел репутаційних даних, оптимізації алгоритмів обробки та підвищення автоматизації процесів аналізу.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУЦІЯХ

4.1 Охорона праці

Організація трудового процесу із застосуванням екранних пристроїв, електронно-обчислювальної техніки та персональних комп'ютерів передбачає обов'язкове впровадження внутрішньозмінних регламентованих перерв (згідно з НПАОП 0.00-7.11-12). Це особливо важливо для спеціалістів, які працюють з інформаційними системами, займаються адмініструванням, аналізом даних або розробкою програмного забезпечення, оскільки тривале перебування за комп'ютером створює значне навантаження на зір, нервову систему та опорно-руховий апарат. Регламентовані перерви є необхідною умовою збереження здоров'я працівників, попередження розвитку професійно зумовлених захворювань і забезпечення стабільного рівня працездатності протягом робочого дня. Їх необхідно планувати заздалегідь, до появи ознак перевтоми, як фізіологічного, так і суб'єктивного характеру [21].

У випадку виконання працівником кількох видів діяльності, наприклад опрацювання технічної документації, написання коду, тестування систем або моніторингу мережевої інфраструктури, основною вважається та, що займає щонайменше половину тривалості робочого часу та пов'язана з роботою за екранним пристроєм. Протягом робочої зміни повинні бути передбачені перерви на прийом їжі, короткочасні паузи для задоволення особистих потреб відповідно до чинного трудового законодавства, а також додаткові спеціалізовані перерви, обумовлені специфікою професійної діяльності у сфері ІТ (згідно з НПАОП 0.00-7.14-17).

Залежно від характеру виконуваних завдань працівників, які використовують комп'ютерну техніку, умовно поділяють на три основні групи: розробників програмного забезпечення, операторів електронно-обчислювальних машин та операторів комп'ютерного набору. Кожна з цих категорій характеризується особливими умовами праці, що відрізняються рівнем

інтелектуального навантаження, ступенем напруження зорового аналізатора, інтенсивністю взаємодії з технічними засобами, фізичною активністю та психоемоційним станом. Для фахівців у галузі кібербезпеки, системного адміністрування або моніторингу інформаційних ресурсів додатковим фактором є необхідність підтримання високого рівня концентрації уваги.

Зокрема, діяльність програмістів пов'язана з опрацюванням програмного коду та технічної документації, що потребує тривалої концентрації уваги, супроводжується підвищеним нервово-емоційним напруженням і виконанням роботи у статичній, часто вимушеній позі. Оператори ЕОМ переважно здійснюють введення та обробку інформації, що надходить з комп'ютерної системи або вводиться за відповідним запитом; їх робота зазвичай виконується у відносно вільному темпі, однак також характеризується значним зоровим та емоційним навантаженням. Оператори комп'ютерного набору виконують здебільшого однотипні операції з високою швидкістю введення текстових даних, що зумовлює підвищене навантаження на опорно-руховий апарат кистей рук і органи зору.

Для кожної з зазначених груп рекомендовано встановлювати окремі режими праці та відпочинку за умови восьмигодинної тривалості робочого дня (на основі НПАОП 0.00-7.14-17). Так, для розробників програмного забезпечення рекомендовано запроваджувати перерви тривалістю 15 хвилин після кожної години безперервної роботи; для операторів ЕОМ – після кожних двох годин; для операторів комп'ютерного набору – по 10 хвилин щогодини. У разі неможливості дотримання таких регламентованих перерв тривалість безперервної роботи з екранним пристроєм не повинна перевищувати чотирьох годин. Ці вимоги є однаково актуальними як для офісних працівників, так і для фахівців, які працюють у віддаленому або змішаному режимі [21].

При роботі за дванадцятигодинним графіком упродовж перших восьми часових блоків застосовуються ті самі норми, що й для стандартної зміни. У межах останніх чотирьох годин незалежно від характеру виконуваної діяльності перерви повинні надаватися щогодини тривалістю не менше 15 хвилин. Такий підхід дозволяє мінімізувати ризики перевтоми, втрати концентрації та зниження

продуктивності, що є особливо важливим для фахівців, які працюють з критично важливими інформаційними системами.

З метою зменшення психоемоційного напруження, профілактики зорової втоми, покращення мозкового кровообігу та запобігання негативним наслідкам малорухливого способу роботи доцільно використовувати частину перерв для виконання спеціально розроблених гімнастичних вправ. У межах психофізіологічного розвантаження працівникам також рекомендується застосування елементів саморегуляції, технік розслаблення або короткочасної зміни виду діяльності, що сприяє досягненню емоційної рівноваги та підтриманню високого рівня працездатності під час роботи з комп'ютером.

Сеанси психофізіологічного розвантаження можуть проводитися як у спеціально обладнаних приміщеннях, так і в індивідуальному форматі із використанням засобів аудіовізуальної стимуляції. Вони умовно поділяються на послідовні етапи: адаптації, релаксації та активізації. Навіть короткочасні вправи тривалістю 5–10 хвилин сприяють зменшенню втоми, покращенню самопочуття та підвищенню ефективності трудової діяльності спеціалістів, які працюють у сфері інформаційних технологій.

4.2 Здоровий спосіб життя людини та його вплив на професійну діяльність

Здоров'я людини формується під впливом спадкових особливостей, стилю життя та екологічних умов. Водночас важливу роль відіграє усвідомлене ставлення людини до власного організму та навколишнього середовища. Здоров'я можна розглядати як стан повної соціально-біологічної гармонії, за якого всі органи й системи функціонують узгоджено з природними та соціальними умовами, відсутні хворобливі прояви, психоемоційні порушення та фізичні вади.

Оцінка стану здоров'я ґрунтується на сукупності різних показників, проте в узагальненому вигляді воно визначається як природний стан організму, що характеризується рівновагою та відсутністю виражених патологічних змін.

Необхідно враховувати, що здоров'я залежить від багатьох чинників, які в комплексі формують поняття здорового способу життя. Його основне завдання полягає у формуванні відповідального ставлення до власного здоров'я, розвитку фізичної та психічної культури, загартуванні організму, а також раціональній організації праці й відпочинку.

До ключових складових здорового способу життя належить насамперед спосіб життя, який суттєво впливає на стан здоров'я людини та охоплює кілька взаємопов'язаних аспектів: економічний рівень життя, якість життя з соціологічної точки зору, індивідуальний стиль життя та загальний соціально-економічний устрій.

Не менш важливим є рівень культури особистості. Людина виступає одночасно і суб'єктом, і результатом власної діяльності, а культура в цьому контексті проявляється як усвідомлене ставлення до себе. Проте на практиці багато людей ігнорують вимоги здорового способу життя: порушують режим дня, мають шкідливі звички, нераціонально харчуються. Тому знання про збереження здоров'я повинні стати складовою повсякденної поведінки.

Здоров'я також посідає важливе місце в ієрархії потреб людини. Часто воно поступається матеріальним цінностям, що з часом негативно позначається не лише на самій людині, а й на здоров'ї майбутніх поколінь. Саме тому здоров'я має розглядатися як пріоритетна життєва цінність.

Важливу роль відіграє і мотивація. Більшість людей починає усвідомлювати значення здоров'я лише після його суттєвої втрати. Лише тоді з'являється прагнення подолати захворювання та повернутися до повноцінного життя.

Окрему увагу слід приділяти так званим зворотним зв'язкам організму. Тривале ігнорування шкідливого впливу алкоголю, тютюну чи інших негативних чинників призводить до відстрочених наслідків, коли відмова від шкідливих звичок уже не завжди дає бажаний результат.

Формування установки на довге й активне життя передбачає вміння мобілізувати внутрішні резерви організму для подолання життєвих труднощів та зниження ризику захворювань, що сприяє збереженню працездатності й довголіттю. Важливе значення має й навчання здоровому способу життя, яке

починається в сім'ї та доповнюється санітарною освітою. Загальний фізичний і психічний стан людини значною мірою визначає її здатність своєчасно розпізнавати небезпеку та адекватно реагувати на неї.

Суттєвий вплив на здоров'я має психічний стан. Сучасна людина постійно стикається з чинниками ризику, які негативно впливають на нервову й серцево-судинну системи та знижують опірність організму. У таких умовах виникає стресова реакція – природна відповідь організму на надмірне напруження. Стрес являє собою комплекс неспецифічних біохімічних, фізіологічних і психологічних реакцій, що виникають під впливом сильних подразників.

Повна відсутність стресу є несумісною з життям, оскільки помірне напруження необхідне для активної діяльності людини. Водночас інтенсивний і тривалий стрес може стати причиною розвитку різних захворювань або навіть призвести до летальних наслідків.

Реакція на стресові ситуації значною мірою є індивідуальною. Одні люди легко переносять побутові труднощі, але болісно реагують на сімейні проблеми, інші – навпаки. Особливо високий рівень стресу спостерігається у людей молодого віку, оскільки їхні життєві потреби зазвичай значно вищі.

Поведінка людини в екстремальних умовах відіграє ключову роль у формуванні наслідків стресу. Неправильні дії можуть призвести до паніки, істеричних реакцій або втрати самоконтролю, що значно погіршує ситуацію. Стійкість до стресу залежить від типу нервової системи та темпераменту. Традиційно виділяють чотири типи темпераменту: холеричний, сангвінічний, флегматичний і меланхолійний, кожен з яких має свої особливості реагування на складні умови.

Для зміцнення психічної стійкості рекомендується використовувати фізичну активність, заняття спортом, прогулянки на свіжому повітрі та інші природні чинники. Важливим також є вміння контролювати власні емоції та психоемоційне напруження, зберігаючи врівноваженість навіть у складних обставинах.

З метою відновлення працездатності застосовуються різні психологічні методи, зокрема психотерапія, психопрофілактика та психогігієна. Психотерапія

використовується для зменшення страху й тривоги, психопрофілактика спрямована на розвиток навичок саморегуляції, а психогігієна охоплює гармонійні міжособистісні відносини, комфортні умови життя та повноцінний відпочинок.

Фізичне здоров'я людини залежить як від спадкових факторів, так і від соціально-економічних, гігієнічних та кліматичних умов. Одним із ключових показників фізичного стану є енергетичний потенціал організму, що визначає його здатність до життєдіяльності. Фізичний розвиток оцінюється за антропометричними показниками, такими як ріст, маса тіла, розвиток м'язової системи та форма грудної клітки.

Важливе значення для підтримання здоров'я має раціональне харчування, яке забезпечує нормальний ріст, розвиток і функціонування організму, а також сприяє профілактиці захворювань.

Окреме місце займає загартування організму – систематичний дозований вплив природних факторів, зокрема температури, з метою підвищення стійкості до несприятливих умов. Основою загартування є адаптаційні можливості організму, які дозволяють йому пристосовуватися до змін навколишнього середовища. Особливо ефективним вважається загартування холодом, оскільки воно підвищує теплову продукцію, покращує кровообіг і знижує ризик інфекційних захворювань.

До інших засобів зміцнення здоров'я належать ранкова гімнастика, активний відпочинок на природі, перебування в лісовій, гірській або морській місцевості, що позитивно впливає на фізичний і психоемоційний стан людини.

ВИСНОВКИ

У кваліфікаційній роботі було досліджено проблему безпеки веб-активності користувачів у сучасних операційних системах та проаналізовано підходи до виявлення потенційно небезпечних веб-ресурсів. Метою дослідження було підвищення ефективності ідентифікації шкідливих і фішингових URL-адрес шляхом розробки програмної утиліти моніторингу, що використовує агрегований аналіз даних з кількох незалежних джерел загрозової аналітики.

У ході роботи було обґрунтовано вибір модульної архітектури та технологічного стеку на базі мови програмування Rust, що забезпечує надійність, продуктивність і безпечну роботу з системними ресурсами. Реалізована утиліта інтегрується з сервісами VirusTotal, URLScan.io та AbuseIPDB, дозволяючи отримувати антивірусні, поведінкові та мережеві характеристики веб-ресурсів. Для узагальнення отриманих результатів запропоновано математичну модель інтегральної оцінки ризику, яка формує числовий показник безпеки та дискретний рівень загрози.

Проведене тестування підтвердило коректність роботи розробленого програмного засобу. Система продемонструвала високу точність класифікації безпечних і шкідливих URL-адрес, відсутність хибних спрацювань для довірених доменів та стабільну роботу механізмів кешування, що зменшує навантаження на зовнішні API. Навантажувальні випробування показали мінімальний вплив утиліти на продуктивність системи.

Отримані результати мають практичне значення для фахівців з інформаційної безпеки та можуть бути використані для аудиту веб-активності й підвищення рівня захищеності кінцевих точок. Подальший розвиток роботи вбачається у розширенні джерел аналізу та впровадженні більш проактивних механізмів реагування на загрози.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IBM Security. (2024). Cost of a data breach report 2024. https://www.ibm.com/reports/data-breach.

2. National Institute of Standards and Technology. (n.d.). Data loss prevention overview. https://csrc.nist.gov/glossary/term/data_loss_prevention.

3. National Institute of Standards and Technology. (2017). Guide to intrusion detection and prevention systems (IDPS) (NIST SP 800-94). https://csrc.nist.gov/publications/detail/sp/800-94/final.

4. Lockheed Martin. (n.d.). Cyber Kill Chain®. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

5. VirusTotal. (n.d.). VirusTotal documentation. https://developers.virustotal.com/reference/overview.

6. AbuseIPDB. (n.d.). AbuseIPDB API documentation. https://docs.abuseipdb.com.

7. PhishTank. (n.d.). PhishTank: Join the fight against phishing. https://www.phishtank.com.

8. Spamhaus Project. (n.d.). Spamhaus project – the top threat intelligence provider. https://www.spamhaus.org.

9. MISP Project. (n.d.). MISP – Malware information sharing platform & threat sharing. https://www.misp-project.org.

10. TheHive Project. (n.d.). Incident response platform. https://thehive-project.org.

11. Suricata Foundation. (n.d.). Suricata – open source IDS / IPS / NSM engine. https://suricata.io.

12. Zeek. (n.d.). Zeek network security monitor. https://zeek.org.

13. SQLite Consortium. (n.d.). SQLite documentation. https://www.sqlite.org/docs.html.

14. Mozilla. (n.d.). Mozilla Firefox places database (places.sqlite). https://developer.mozilla.org/en-US/docs/Mozilla/Tech/Places.

15. Klabnik, S., & Nichols, C. (n.d.). The Rust programming language. https://doc.rust-lang.org/book.

16. Rust Foundation. (n.d.). Rust security guarantees. https://www.rust-lang.org.

17. OWASP Foundation. (n.d.). OWASP top 10 web application security risks. https://owasp.org/www-project-top-ten/.

18. MITRE. (n.d.). MITRE ATT&CK® framework. https://attack.mitre.org.

19. JSON Lines. (n.d.). JSON Lines documentation. https://jsonlines.org.

20. Linux Audit. (n.d.). Linux audit framework documentation. https://linux-audit.com.

21. Міністерство соціальної політики України. (2018). Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями: Наказ № 207 від 14.02.2018. Верховна Рада України. https://zakon.rada.gov.ua/go/z0508-18.

22. Лапін В. М**.** Безпека життєдіяльності людини : навч. посібник. – Львів : Львівський банківський коледж, 1998. – 192 с..

23. Skorenkyy Y., Kozak R., Zagorodna N., Kramar O., Baran, I. Use of augmented reality-enabled prototyping of cyber-physical systems for improving cybersecurity education. Journal of Physics: Conference Series. 2021. Vol. 1840, No. 1. DOI: http://dx.doi.org/10.1088/1742-6596/1840/1/012026.

24. Lypa B., Horyn I., Zagorodna N., Tymoshchuk D., Lechachenko T. Comparison of feature extraction tools for network traffic data. CEUR Workshop Proceedings. 2024. vol. 3896. P. 1-11.

25. ТИМОЩУК Д., ЯЦКІВ В. USING HYPERVISORS TO CREATE A CYBER POLYGON. MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES, 2024, (3). P. 52-56. https://doi.org/10.31891/2219-9365-2024-79-7

26. Kharchenko A., Bodnarchuk I., Yatcysyn V. The Method for Comparative Evaluation of Software Architecture with Accounting of Trade-offs. American Journal of Information Systems. 2014. Vol. 2, No. 1. P. 20-25. URL: http://pubs.sciepub.com/ajis/2/1/5/.

28. Shmatko O., Balakireva S., Vlasov A. et al. Development of methodological foundations for designing a classifier of threats to cyberphysical systems. Eastern-European Journal of Enterprise Technologies. 2020. 3/9 (105). P. 6-19.

29. Karpinski M., Korchenko A., Vikulov P., Kochan R. The Etalon Models of Linguistic Variables for Sniffing-Attack Detection, in Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2017 IEEE 9th International Conference on, 2017, pp. 258-264.

30. Микитишин А. Г., Митник М. М., Стухляк П. Д. Телекомунікаційні системи та мережі. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2017. 384 с.

31. Stadnyk, M., Fryz, M., & Scherbak, L. The Feature Extraction and Estimation of a Steady-state Visual Evoked Potential by the Karhunen-Loeve Expansion. *Eastern-European Journal of Enterprise Technologies*, *1*(4), 56-62.

32. Revniuk, O., Zagorodna, N., Kozak, R., & Yavorskyy, B. (2025). Development of an information system for the quantitative assessment of web application security based on the OWASP ASVS standard. *Вісник Тернопільського національного технічного університету*, *118*(2), 56-65.

33. Lupenko, S., Orobchuk, O., Kateryniuk, I., Kozak, R., & Lypak, H. (2024). Secure information system for Chinese Image medicine knowledge consolidation. In *ITTAP* (pp. 509-519).

34. Nedzelky, D., Derkach, M., Skarga-Bandurova, I., Shumova, L., Safonova, S., & Kardashuk, V. (2021, September). A Load Factor and its Impact on the Performance of a Multicore System with Shared Memory. In 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) (Vol. 1, pp. 499-503). IEEE.

35. Kozak, R., Skorenkyy, Y., Kramar, O., Lechachenko, T., & Brevus, H. (2025). Cybersecurity provisioning for Industry 4.0 digital twin with AR components. In *CEUR Workshop Proceedings, 3rd International Workshop on Computer Information Technologies in Industry* (Vol. 4, pp. 166-178).

ДОДАТОК А Публікація

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.056.53
О. Семчишин, студент; Р. Козак, к.т.н., доцент
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

МЕТОДІВ АНАЛІЗУ ШКІДЛИВИХ ПОСИЛАНЬ ТА IP-АДРЕС У СЕРЕДОВИЩІ LINUX

UDC 004.056.53
O. Semchyshyn, student; R. Kozak, Ph.D., Assoc. Prof.

METHODS FOR ANALYZING MALICIOUS LINKS AND IP ADDRESSES IN THE LINUX ENVIRONMENT

Сучасні кіберзагрози активно використовують шкідливі URL-посилання та IP-адреси для організації фішингових кампаній, доставки шкідливого програмного забезпечення та підтримання взаємодії з командно-керувальними серверами. Тому важливим завданням є оперативний аналіз таких індикаторів компрометації (IoC) у середовищі Linux, яке широко застосовується для серверів, систем моніторингу та інструментів кіберзахисту[1].

Одним із базових методів є репутаційний аналіз URL та IP через зовнішні сервіси: VirusTotal, AbuseIPDB, URLScan. У середовищі Linux такі сервіси можуть бути інтегровані через API, що дозволяє автоматизувати масову перевірку індикаторів засобами Bash або Python. Використання інструментів curl, jq та системи черг дозволяє формувати конвеєр обробки, у якому кожен елемент проходить оцінку за рівнем ризику.

Другим важливим етапом є локальний динамічний аналіз, що не залежить від зовнішніх платформ. До нього належать інструменти:

- wget – для завантаження та аналізу HTML-вмісту;
- curl -I – для перевірки редиректів та статус-кодів;
- dig та nslookup – для аналізу DNS-структури домену;
- whois – для визначення реєстратора, AS-номеру та країни розміщення IP.

Такі інструменти дають змогу виявити підроблені домени, ознаки фішингу, нестандартні DNS-записи, а також домени– «одноразівки», реєстровані спеціально для атак [2, 3].

Третім етапом є статичний аналіз та зіставлення з існуючими IOC-базами. У Linux широко застосовуються yara, threatfox-cli, локальні списки IP-адрес та URL із MalwareBazaar, ThreatFox, OpenPhish. Це дозволяє створювати власні правила виявлення підозрілих структур, аналізувати шаблони вмісту та виявляти повторювані ознаки шкідливої активності[3].

Передбачається, що поєднання репутаційного, динамічного та статичного аналізу може забезпечити помітно вищу ефективність класифікації шкідливих URL та IP-адрес порівняно з використанням лише зовнішніх сервісів. Такий комплексний підхід, імовірно, дає змогу точніше виявляти підозрілу активність завдяки багаторівневій перевірці індикаторів. DLP системи захищають конфіденційні дані від витоків, контролюючи канали їх передачі та використання. Це дозволяє організаціям та бізнесу мінімізувати ризики фінансових втрат, репутаційних ударів і юридичних проблем.

Література

1. Almomani A., Gupta B., Atawneh S., Meulenberg A. The Scope of Phishing Attacks: A Comprehensive Study. IEEE Communications Surveys & Tutorials. 2013. 29 p.
2. Malicious URLs and IP Address Threat Analysis Report. Palo Alto Networks Unit 42. URL: https://www.content.shi.com/cms-content/accelerator/media/pdfs/palo-alto/palo-alto-122623-unit42-threat-report.pdf (дата звернення: 1.12.2025).
3. A Project from abuse.ch for Malware Samples and IOC Feeds. MalwareBazaar URL: https://bazaar.abuse.ch/statistics/ (дата звернення: 3.12.2024).

ДОДАТОК Б Код реалізованого веб інтерфейсу подій

Лістинг коду програми app.py

```
#!/usr/bin/env python3
from flask import Flask, render_template, jsonify, request
from datetime import datetime, timedelta, timezone
import json
import os
from pathlib import Path
from collections import defaultdict

app = Flask(__name__)
app.config['SECRET_KEY'] = 'firefox-monitor-soar-2025'


LOGS_DIR = Path(__file__).parent / 'logs'
EVENTS_LOG = LOGS_DIR / 'security_events.jsonl'
PROCESSED_LOG = LOGS_DIR / 'processed_events.jsonl'

def ensure_logs_dir():
    LOGS_DIR.mkdir(exist_ok=True)
    if not EVENTS_LOG.exists():
        EVENTS_LOG.touch()

def parse_log_line(line):
    try:
        return json.loads(line.strip())
    except:
        return None

def load_events(hours=24, severity_filter=None):
    ensure_logs_dir()

    events = []
    cutoff_time = datetime.now(timezone.utc) -
timedelta(hours=hours)

    try:
        with open(EVENTS_LOG, 'r', encoding='utf-8') as f:
            for line in f:
                event = parse_log_line(line)
                if not event:
                    continue
                try:
                    event_time =
datetime.fromisoformat(event.get('timestamp', ''))
                    if event_time.tzinfo is None:
                        event_time =
event_time.replace(tzinfo=timezone.utc)
                    if event_time < cutoff_time:
                        continue
                except (ValueError, AttributeError):
```

```
                    continue
                if severity_filter:
                    if event.get('severity') not in
severity_filter:
                        continue

                events.append(event)
    except FileNotFoundError:
        pass

    events.sort(key=lambda x: x.get('timestamp', ''),
reverse=True)

    return events

def get_statistics(hours=24):
    events = load_events(hours=hours)

    stats = {
        'total': len(events),
        'high': sum(1 for e in events if e.get('severity') ==
'High'),
        'medium': sum(1 for e in events if e.get('severity') ==
'Medium'),
        'low': sum(1 for e in events if e.get('severity') ==
'Low'),
        'blocked': sum(1 for e in events if e.get('action') ==
'blocked'),
        'allowed': sum(1 for e in events if e.get('action') ==
'allowed'),
    }

    threats = defaultdict(int)
    for event in events:
        if event.get('severity') in ['High', 'Medium']:
            domain = event.get('domain', 'Unknown')
            threats[domain] += 1

    stats['top_threats'] = sorted(threats.items(), key=lambda x:
x[1], reverse=True)[:5]

    return stats

@app.route('/')
def index():
    stats = get_statistics(hours=24)
    return render_template('dashboard.html', stats=stats)

@app.route('/api/events')
def api_events():
    hours = int(request.args.get('hours', 24))
    severity = request.args.get('severity', 'Medium,High')
```

```
    severity_filter = severity.split(',') if severity else None
    events = load_events(hours=hours,
severity_filter=severity_filter)

    return jsonify({
        'success': True,
        'count': len(events),
        'events': events
    })

@app.route('/api/stats')
def api_stats():
    hours = int(request.args.get('hours', 24))
    stats = get_statistics(hours=hours)

    return jsonify({
        'success': True,
        'stats': stats
    })

@app.route('/events')
def events_page():
    return render_template('events.html')

@app.route('/api/event/<event_id>')
def api_event_detail(event_id):
    events = load_events(hours=168)  # 7 днів

    for event in events:
        if event.get('id') == event_id:
            return jsonify({
                'success': True,
                'event': event
            })

    return jsonify({
        'success': False,
        'error': 'Event not found'
    }), 404

@app.route('/api/event/<event_id>/process', methods=['POST'])
def process_event(event_id):
    ensure_logs_dir()

    try:
        events = []
        processed_event = None

        if not EVENTS_LOG.exists():
            return jsonify({
                'success': False,
                'error': 'Events log not found'
            }), 404
```

```
        with open(EVENTS_LOG, 'r', encoding='utf-8') as f:
            for line in f:
                event = parse_log_line(line)
                if event and event.get('id') == event_id:
                    processed_event = event
                    processed_event['processed_at'] =
datetime.now(timezone.utc).isoformat()
                    processed_event['processed'] = True
                else:
                    if event:
                        events.append(line)

        if not processed_event:
            return jsonify({
                'success': False,
                'error': 'Event not found'
            }), 404

        with open(PROCESSED_LOG, 'a', encoding='utf-8') as f:
            f.write(json.dumps(processed_event) + '\n')

        with open(EVENTS_LOG, 'w', encoding='utf-8') as f:
            f.writelines(events)

        return jsonify({
            'success': True,
            'message': 'Event processed successfully',
            'event_id': event_id
        })

    except Exception as e:
        return jsonify({
            'success': False,
            'error': str(e)
        }), 500

@app.route('/health')
def health():
    return jsonify({
        'status': 'healthy',
        'timestamp': datetime.now(timezone.utc).isoformat()
    })

if __name__ == '__main__':
    ensure_logs_dir()
    print("SOAR Dashboard запущено")
    print("Dashboard: http://localhost:5000")
    print("Events: http://localhost:5000/events")
    print("API: http://localhost:5000/api/events")
    app.run(host='0.0.0.0', port=5000, debug=True)
```