**ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО**
**«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»**
**Інститут комп'ютерно-інформаційних технологій та дизайну**
**Кафедра комп'ютерних інформаціних систем та технологій**

**Стрепков Владислав Олександрович**

**Дипломна робота**
**на здобуття ступеня бакалавра**
на тему:

# «Розробка системи аналізу логів вебсервера для виявлення потенційних кібератак»

Група ІК-922Б1ІПЗ*(4.0д)
Галузь знань: (F) Інформаційні технології
Спеціальність: F2 (121) «Інженерія програмного забезпечення»
**Рівень вищої освіти: перший (бакалаврский)**
*Дипломна робота містить результати власних досліджень. Використання ідей, результатів, текстів інших авторів мають посилання на відповідне джерело*

_____ Стрепков В. О.
підпис

Науковий керівник

голова Наукового товариства студентів та аспірантів. _____ Яременко Д. М.
(посада, науковий ступінь, вчене звання) підпис (прізвище та ініціали)

Завідувач кафедри _____ Кавун С. В.
підпис (прізвище та ініціали)

Професор кафедри, д.е.н. професор
(посада, науковий ступінь, вчене звання)

Київ 2026 р.

# ЗМІСТ

# ВСТУП

**Актуальність теми.** У сучасних умовах стрімкого розвитку цифрових технологій, активного використання вебсервісів, електронних платформ, хмарних інфраструктур та автоматизованих інформаційних систем питання забезпечення кібербезпеки набуває особливої актуальності. Вебсервери є одним із ключових елементів інформаційної інфраструктури організацій, підприємств, державних установ, освітніх закладів та комерційних сервісів, оскільки саме через них здійснюється доступ користувачів до вебресурсів, обробка запитів, передача даних, авторизація, взаємодія з базами даних та іншими компонентами інформаційної системи. Водночас відкритість вебсервера до мережевого середовища робить його одним із найпоширеніших об’єктів для здійснення кібератак, несанкціонованого доступу, автоматизованого сканування вразливостей, спроб підбору облікових даних, SQL-ін’єкцій, XSS-атак, DDoS-активності, звернень до заборонених ресурсів, експлуатації помилок конфігурації та інших форм підозрілої активності.

Особливе значення у процесі виявлення потенційних кібератак має аналіз логів вебсервера, оскільки журнали подій містять детальну інформацію про IP-адреси користувачів, час запитів, HTTP-методи, URL-адреси, коди відповідей сервера, обсяг переданих даних, User-Agent, частоту звернень та інші параметри, які можуть свідчити про нормальну або аномальну поведінку. Саме логи вебсервера дають змогу відстежувати динаміку запитів, виявляти нетипові шаблони поведінки, аналізувати повторювані помилки авторизації, фіксувати спроби доступу до адміністративних сторінок, визначати джерела підозрілих звернень та своєчасно реагувати на потенційні загрози. У практичному аспекті така інформація є важливою не лише для адміністраторів вебресурсів, а й для фахівців із кібербезпеки, які здійснюють моніторинг, розслідування інцидентів та побудову системи превентивного захисту.

Актуальність теми також зумовлена тим, що традиційний ручний аналіз серверних логів є складним, трудомістким і малоефективним у разі великого обсягу даних. У реальних умовах вебсервер може генерувати тисячі або навіть

мільйони записів журналу подій, що унеможливлює оперативне виявлення загроз без використання автоматизованих засобів обробки, фільтрації, класифікації та візуалізації даних. Тому розробка системи аналізу логів вебсервера для виявлення потенційних кібератак є важливим практичним завданням, спрямованим на підвищення рівня захищеності вебресурсів, зменшення часу реагування на інциденти та формування більш ефективного механізму контролю користувацької активності.

**Об’єктом дослідження** є процес моніторингу та аналізу подій вебсервера в контексті виявлення підозрілої мережевої активності та потенційних кібератак.

**Предметом дослідження** є методи, алгоритми та програмні засоби обробки логів вебсервера, що використовуються для виявлення аномальних запитів, підозрілих шаблонів поведінки користувачів і можливих ознак кіберзагроз.

**Метою роботи** є розробка системи аналізу логів вебсервера, яка забезпечує автоматизовану обробку журналів подій, виявлення потенційно небезпечної активності, формування оцінки ризику та відображення результатів моніторингу у зручному для адміністратора вигляді.

**Для досягнення поставленої мети сформульовано такі завдання:**

дослідити сутність, призначення та структуру логів вебсервера як джерела інформації для виявлення кіберзагроз;

проаналізувати основні види кібератак на вебсервери та характерні ознаки їх прояву в журналах подій;

розглянути існуючі підходи до аналізу серверних логів, виявлення аномальної активності та оцінювання рівня ризику користувацьких запитів;

визначити функціональні вимоги до системи аналізу логів вебсервера;

розробити алгоритм обробки логів та виявлення потенційно підозрілої активності на основі визначених критеріїв;

реалізувати програмні компоненти системи, зокрема механізми завантаження, парсингу, фільтрації, аналізу та збереження даних;

створити адміністративну панель для перегляду результатів аналізу, журналу подій і візуалізації показників підозрілої активності;

провести тестування розробленої системи на змодельованих сценаріях потенційних кібератак;

оцінити ефективність роботи системи та визначити можливі напрями її подальшого вдосконалення.

Методологічну основу дослідження становить сукупність загальнонаукових, аналітичних і прикладних методів, які забезпечили комплексне вивчення проблеми розробки системи аналізу логів вебсервера. Метод аналізу було використано для дослідження структури серверних журналів, основних типів HTTP-запитів, кодів відповідей сервера та характерних ознак підозрілої активності. Метод порівняння дав змогу зіставити нормальні та аномальні шаблони поведінки користувачів, а також визначити відмінності між типовими помилками роботи вебресурсу та потенційними ознаками кібератак. Метод узагальнення застосовано для формування критеріїв оцінювання ризику, які можуть використовуватися під час автоматизованого аналізу логів.

У процесі розробки програмної частини роботи було використано метод моделювання, що дозволив створити умовні сценарії підозрілої активності, зокрема часті запити з однієї IP-адреси, спроби доступу до адміністративних сторінок, звернення до неіснуючих ресурсів, підозрілі параметри в URL та інші потенційно небезпечні дії. Також застосовувалися методи алгоритмізації та програмної реалізації, за допомогою яких було побудовано логіку обробки записів журналу, класифікації подій, визначення рівня ризику та відображення результатів у системі. Для перевірки працездатності розробленого рішення було використано метод тестування, що дав змогу оцінити коректність роботи основних функціональних компонентів системи.

**Практичне значення** роботи полягає у можливості використання розробленої системи як допоміжного інструменту для адміністраторів вебресурсів, фахівців із кібербезпеки та розробників вебсистем з метою своєчасного виявлення потенційно небезпечної активності на основі аналізу логів вебсервера. Запропоноване рішення може бути застосоване для автоматизації первинного моніторингу журналів подій, зменшення навантаження на

адміністратора, швидкого виявлення підозрілих IP-адрес, аналізу частоти запитів, фіксації помилок доступу та формування загального уявлення про рівень ризику користувацької активності.

Результати роботи можуть бути використані у навчальних цілях під час вивчення дисциплін, пов’язаних із кібербезпекою, веброзробкою, адмініструванням серверів, захистом інформаційних систем та аналізом даних. Крім того, розроблена система може бути вдосконалена шляхом інтеграції з реальними вебсерверами, розширенням переліку правил виявлення атак, додаванням механізмів машинного навчання, автоматичного блокування підозрілих IP-адрес або формування сповіщень для адміністратора.

Структурно робота складається зі вступу, трьох розділів, висновків, списку використаних джерел. У вступі обґрунтовано актуальність теми, визначено об’єкт, предмет, мету, завдання, методологію дослідження, практичне значення та структуру роботи.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ АНАЛІЗУ ЛОГІВ ВЕБСЕРВЕРА В СИСТЕМІ КІБЕРБЕЗПЕКИ

## 1.1. Поняття, структура та призначення логів вебсервера

У сучасній вебінфраструктурі логи вебсервера є одним із базових джерел технічної, діагностичної та безпекової інформації про функціонування вебресурсу. Кожне звернення користувача або автоматизованого клієнта до вебсервера залишає певний інформаційний слід, який може бути зафіксований у журналі подій. Такий запис відображає не лише факт доступу до певного ресурсу, а й низку супровідних параметрів, що мають значення для адміністрування, оптимізації продуктивності, розслідування інцидентів та виявлення потенційних кібератак. У найбільш загальному розумінні логи вебсервера можна розглядати як упорядковану сукупність записів про HTTP-запити, відповіді сервера, помилки обробки, службові події та інші технічні стани, які виникають під час взаємодії клієнта з вебсистемою. Саме тому журнали подій є важливим елементом спостережності програмної системи, оскільки дають змогу не тільки зафіксувати факт події, а й відновити її контекст, встановити часову послідовність дій, виявити джерело звернення та оцінити характер активності [1].

З технічної точки зору вебсервер виконує роль посередника між клієнтським програмним середовищем, наприклад браузером, скриптом, ботом або API-клієнтом, і ресурсами вебсайту чи вебзастосунку. Під час обробки запиту сервер приймає HTTP-метод, визначає запитуваний шлях, перевіряє доступність ресурсу, формує відповідь, повертає відповідний код стану та, залежно від конфігурації, записує інформацію про цю взаємодію до лог-файлу. У класичних конфігураціях Apache HTTP Server та Nginx найбільш поширеними є журнали доступу й журнали помилок. Журнал доступу фіксує факти звернення до ресурсу, а журнал помилок містить інформацію про помилки обробки, проблеми конфігурації, збої модулів, некоректні запити або інші події, які впливають на роботу вебсервера [2]. У контексті теми цієї роботи особливо важливим є саме журнал доступу, оскільки в ньому накопичуються дані, придатні для виявлення аномальної активності: частих звернень з однієї IP-адреси, сканування директорій,

спроб доступу до адміністративних сторінок, підозрілих параметрів у URL, великої кількості помилок 404 або 403, використання нетипових User-Agent та інших ознак потенційної атаки.

Структура логів вебсервера залежить від конкретного програмного забезпечення, його конфігурації та обраного формату запису. У типовому журналі доступу можуть міститися IP-адреса клієнта, дата і час запиту, HTTP-метод, запитуваний ресурс, версія протоколу, код відповіді сервера, обсяг переданих даних, адреса сторінки-джерела переходу, рядок User-Agent, а також додаткові службові поля. Наприклад, запис у форматі combined log format може містити відомості про віддалений хост, ідентифікаційні дані користувача, часову мітку, повний рядок запиту, статус відповіді, кількість переданих байтів, referer та User-Agent. Кожне з цих полів має окреме аналітичне значення. IP-адреса дозволяє групувати запити за джерелом, часова мітка дає змогу визначати інтенсивність активності, HTTP-метод показує характер дії, URL відображає об’єкт звернення, код відповіді вказує на результат обробки запиту, а User-Agent може допомогти розрізнити звичайний браузер, пошукового робота, сканер вразливостей або підроблений клієнтський ідентифікатор [3].

У межах аналізу логів особливу роль відіграють HTTP-методи, оскільки вони дають уявлення про тип взаємодії клієнта з вебсервером. Найпоширенішими є GET і POST, які використовуються для отримання ресурсів і надсилання даних. Водночас у логах можуть траплятися методи HEAD, PUT, DELETE, OPTIONS, PATCH та інші, що в окремих випадках може мати діагностичне або безпекове значення. Наприклад, неочікувані запити з методами PUT або DELETE до публічного вебресурсу можуть свідчити про спробу перевірки можливості несанкціонованої зміни або видалення даних, якщо такі методи не передбачені логікою застосунку. Так само часті OPTIONS-запити можуть бути пов’язані з технічною перевіркою підтримуваних сервером методів. Сам по собі HTTP-метод не є доказом атаки, однак у поєднанні з іншими параметрами, такими як IP-адреса, частота звернень, код відповіді та шлях ресурсу, він утворює важливу ознаку для автоматизованого аналізу поведінки користувачів.

Не менш важливим елементом структури логів є коди стану HTTP, які відображають результат обробки запиту сервером. Коди групи 2xx свідчать про успішну обробку, 3xx пов’язані з перенаправленнями, 4xx вказують на помилки з боку клієнта, а 5xx свідчать про помилки на стороні сервера. Для системи виявлення потенційних кібератак особливе значення мають коди 400, 401, 403, 404, 405, 429, 500 та 503. Велика кількість відповідей 404 може свідчити про сканування неіснуючих файлів і директорій, часті відповіді 401 або 403 можуть бути пов’язані зі спробами несанкціонованого доступу, код 405 може вказувати на використання непідтримуваних методів, а 429 може свідчити про перевищення допустимої частоти запитів. Помилки 5xx не завжди є наслідком атаки, однак їхнє різке зростання може вказувати на перевантаження, некоректні запити, збої застосунку або спробу викликати нестабільну поведінку системи [4].

Часова мітка є одним із ключових параметрів логів, оскільки саме вона дозволяє перейти від простого переліку подій до аналізу динаміки активності. Без урахування часу окремий запит може не мати суттєвого значення, однак серія однотипних запитів за короткий проміжок може свідчити про автоматизоване сканування, brute force-активність, спробу перевантаження вебресурсу або інший підозрілий сценарій. Тому під час розробки системи аналізу логів важливо передбачити можливість групування подій за часовими інтервалами, визначення кількості запитів на хвилину або секунду, виявлення пікових навантажень, порівняння активності різних IP-адрес та формування часової картини інциденту. Саме часова послідовність дозволяє відокремити випадкові помилки користувача від системної підозрілої активності, оскільки атаки на вебсервери часто мають повторюваний, масовий або автоматизований характер.

IP-адреса клієнта є ще одним важливим структурним елементом логів, однак її аналітичне значення потребує обережного тлумачення. З одного боку, IP-адреса дає змогу ідентифікувати джерело запитів, групувати події, будувати рейтинг активних клієнтів, виявляти надмірну кількість звернень або повторювані спроби доступу до заборонених ресурсів. З іншого боку, IP-адреса не завжди однозначно відповідає конкретній особі або пристрою, оскільки користувачі можуть

перебувати за NAT, використовувати VPN, проксі-сервери, мобільні мережі або інші проміжні інфраструктурні рішення. Крім того, у разі роботи вебресурсу за балансувальником навантаження або CDN фактична IP-адреса клієнта може передаватися через спеціальні заголовки, а не безпосередньо відображатися в стандартному полі журналу. Тому для коректного аналізу система повинна враховувати конфігурацію інфраструктури та не зводити оцінку ризику лише до IP-адреси, а використовувати її в комплексі з іншими ознаками.

URL-адреса або шлях запитуваного ресурсу має особливе значення для виявлення кібератак, оскільки саме в цьому полі часто проявляються спроби експлуатації вразливостей. У логах можуть фіксуватися звернення до типових адміністративних сторінок, неіснуючих конфігураційних файлів, резервних копій, службових директорій, файлів на кшталт .env, phpinfo.php, wp-login.php, admin, backup, config, database та інших ресурсів, які часто перевіряються автоматизованими сканерами. Також у URL можуть міститися підозрілі параметри, пов’язані з SQL-ін’єкціями, XSS-атаками, path traversal, command injection або іншими типами атак на вебзастосунки. Наприклад, наявність у параметрах запиту фрагментів union select, script, ../, passwd, cmd, exec або нетипових закодованих символів не завжди автоматично підтверджує атаку, але є достатньою підставою для підвищення рівня ризику події. Тому поле URL є центральним для побудови правил виявлення підозрілої активності в системі аналізу логів [5].

Окрему роль у структурі логів відіграє User-Agent, який містить інформацію про клієнтське програмне середовище. У звичайних умовах це може бути браузер користувача, мобільний застосунок, пошуковий робот або інший легітимний клієнт. Водночас у практиці кібербезпеки User-Agent часто використовується як додаткова ознака для виявлення автоматизованих інструментів, сканерів, ботів або нестандартних клієнтів. Підозру можуть викликати порожні User-Agent, надто короткі або явно технічні рядки, повторювані шаблони, назви відомих утиліт для тестування вебресурсів, а також User-Agent, які не відповідають характеру запитів. Разом із тим це поле легко підробити, тому його не можна розглядати як

самостійну підставу для висновку про атаку. Його значення полягає у посиленні або уточненні загальної оцінки ризику, особливо коли воно поєднується з високою частотою запитів, підозрілими URL, помилками доступу або іншими аномальними показниками.

Призначення логів вебсервера не обмежується лише збереженням історії звернень. У практичному аспекті вони виконують кілька взаємопов'язаних функцій: діагностичну, аналітичну, безпекову, доказову та управлінську. Діагностична функція полягає у виявленні технічних помилок, збоїв, неправильної конфігурації, недоступності ресурсів або проблем із продуктивністю. Аналітична функція пов'язана з оцінкою поведінки користувачів, популярності сторінок, навантаження на сервер і загальної динаміки трафіку. Безпекова функція полягає у виявленні підозрілої активності, ознак атак, несанкціонованих спроб доступу та інших загроз. Доказова функція проявляється в тому, що логи можуть використовуватися для відновлення подій під час розслідування інциденту. Управлінська функція полягає у використанні результатів аналізу для прийняття рішень щодо налаштування правил доступу, оптимізації архітектури, посилення захисту або зміни політик моніторингу.

У системах кібербезпеки логи вебсервера мають особливе значення через їхню наближеність до реальної поведінки користувачів і потенційних зловмисників. На відміну від абстрактних моделей загроз, журнали подій відображають фактичні спроби взаємодії з вебресурсом. Саме в них можна побачити, які сторінки перевірялися, які параметри передавалися, з яких адрес надходили запити, які помилки виникали, які ресурси були недоступними, коли відбувалися пікові навантаження та які дії повторювалися. Це робить логи цінним джерелом для побудови систем виявлення аномалій, формування правил кореляції подій, створення панелей моніторингу та оцінювання рівня ризику. У цьому контексті важливим є підхід OWASP, згідно з яким логування у вебзастосунках має бути не випадковим набором технічних записів, а продуманим механізмом фіксації безпеково значущих подій, який не повинен розкривати зайву конфіденційну інформацію та має бути захищений від несанкціонованого доступу [6].

Разом із тим логи вебсервера мають певні обмеження, які необхідно враховувати під час розробки системи їх аналізу. Вони не завжди містять повний контекст дії користувача, можуть бути неповними через неправильну конфігурацію, можуть не включати ті заголовки або параметри, які важливі для конкретного сценарію аналізу. Крім того, велика кількість записів ускладнює ручну обробку, а різні вебсервери, проксі-сервери, фреймворки та застосунки можуть використовувати відмінні формати логування. У великих системах журнали можуть генеруватися не лише вебсервером, а й балансувальниками навантаження, контейнерами, хмарними сервісами, базами даних, WAF, системами автентифікації та прикладними модулями. Це створює проблему неоднорідності даних, через яку виникає потреба у парсингу, нормалізації, уніфікації часових форматів, очищенні даних і приведенні записів до структури, придатної для автоматизованого аналізу [7].

У межах програмної реалізації системи аналізу логів важливо розглядати лог не лише як текстовий рядок, а як джерело структурованих ознак. Першим етапом обробки є парсинг, тобто виділення з рядка окремих полів: IP-адреси, часу, методу, URL, коду відповіді, розміру відповіді, User-Agent та інших параметрів. Після цього записи можуть проходити нормалізацію, під час якої часові мітки переводяться до єдиного формату, URL очищуються або декодуються, коди відповідей групуються за категоріями, а IP-адреси агрегуються для подальшого аналізу. Наступним етапом може бути фільтрація технічного шуму, наприклад звернень до статичних файлів, зображень, стилів або скриптів, якщо вони не мають значення для конкретного сценарію виявлення атак. Після цього система може застосовувати правила ризику, евристичні критерії або прості статистичні показники, що дозволяють виявляти підозрілі запити та групувати їх у потенційні інциденти.

З позиції інженерії програмного забезпечення логування має бути враховане ще на етапі проєктування системи, а не додаватися лише після виникнення проблеми. Це пов’язано з тим, що якість майбутнього аналізу безпосередньо залежить від того, які саме події фіксуються, в якому форматі вони зберігаються,

наскільки точно визначено часові мітки, чи передбачено збереження контексту події, чи не потрапляють до журналів зайві персональні або конфіденційні дані. У сучасних підходах до безпечного життєвого циклу розроблення програмного забезпечення логування розглядається як елемент контролю, спостережності й подальшого реагування на інциденти, що узгоджується з ідеєю інтеграції вимог кібербезпеки в SDLC та DevSecOps-процеси [8]. Для досліджуваної роботи це означає, що система аналізу логів повинна не просто читати готовий файл, а забезпечувати логічно обґрунтовану обробку даних, яка може бути розширена й адаптована до різних форматів журналів.

У наукових і прикладних дослідженнях логування також пов’язується з поняттям спостережності програмних систем. Спостережність означає здатність системи надавати достатній обсяг даних для розуміння її внутрішнього стану на основі зовнішніх проявів, зокрема логів, метрик і трасування. Для вебсервера це особливо важливо, оскільки адміністратор не може безпосередньо бачити всі процеси взаємодії клієнтів із системою, але може відновлювати їх через журнали подій. Українські дослідники І. Супруненко та В. Рудницький у роботі, присвяченій хмарно-орієнтованій архітектурі адаптивного логування, акцентують увагу на зв’язку логування, спостережності, кібербезпеки та архітектурної моделі програмної системи [9]. Це має безпосереднє значення для теми даної роботи, оскільки система аналізу логів вебсервера фактично є прикладом інструменту, що перетворює технічні записи на зрозумілу інформацію для прийняття рішень.

Важливим аспектом є також питання збереження, захисту та цілісності логів. Якщо журнали подій використовуються для аналізу кіберінцидентів, вони самі стають важливим об’єктом захисту. Потенційний зловмисник може бути зацікавлений у видаленні, зміні або підміні записів, щоб приховати сліди атаки. Тому у практиці адміністрування доцільно передбачати обмеження доступу до лог-файлів, розмежування прав користувачів, резервне копіювання, централізоване збирання журналів, контроль цілісності, ротацію логів і захист від переповнення дискового простору. Надмірне або неконтрольоване логування також може створити ризики, оскільки великі обсяги журналів ускладнюють

аналіз, збільшують навантаження на систему та можуть містити чутливі дані. Тому якісна система аналізу логів повинна враховувати баланс між повнотою фіксації подій, продуктивністю, захистом даних і практичною придатністю журналів для подальшого аналізу.

У контексті виявлення потенційних кібератак логи вебсервера мають не лише інформаційне, а й прогностичне значення. На основі історичних даних можна визначати типову поведінку користувачів, середню кількість запитів, нормальні часові інтервали активності, характерні маршрути переходів, очікувані коди відповідей і звичайні User-Agent. Відхилення від такої базової поведінки можуть свідчити про аномалії. Наприклад, якщо одна IP-адреса протягом короткого часу надсилає сотні запитів до різних неіснуючих сторінок, це може вказувати на сканування. Якщо у URL систематично з’являються фрагменти, характерні для ін’єкцій, це може свідчити про спроби експлуатації вразливостей. Якщо багато запитів завершуються помилками авторизації, це може бути ознакою підбору облікових даних. Якщо різко зростає кількість запитів до ресурсу, це може свідчити про навантажувальну або DDoS-подібну активність. Такі висновки не повинні робитися ізольовано, але вони можуть формувати основу для автоматизованої оцінки рівня ризику [10].

Отже, логи вебсервера є важливим джерелом даних для забезпечення кібербезпеки, оскільки вони фіксують фактичну взаємодію клієнтів із вебресурсом і дають змогу аналізувати поведінку користувачів, технічний стан сервера та потенційні ознаки атак. Їхня структура охоплює низку параметрів, серед яких IP-адреса, час запиту, HTTP-метод, URL, код відповіді, обсяг переданих даних і User-Agent мають найбільше значення для виявлення підозрілої активності. Призначення логів полягає не лише у технічній діагностиці, а й у створенні інформаційної основи для моніторингу, аналізу ризиків, реагування на інциденти та подальшого вдосконалення захисту вебсистем. Саме тому розробка системи аналізу логів вебсервера є логічним і практично значущим напрямом, який поєднує інженерію програмного забезпечення, адміністрування вебінфраструктури та прикладні завдання кібербезпеки.

### 1.2. Основні види кібератак на вебсервери та їх ознаки в журналах подій

Вебсервер є одним із найважливіших елементів інформаційної інфраструктури, оскільки саме через нього забезпечується доступ користувачів до вебсайтів, вебзастосунків, API-сервісів, адміністративних панелей, електронних кабінетів, баз даних та інших цифрових ресурсів. Водночас така відкритість робить вебсервер постійною ціллю для кібератак, які можуть бути спрямовані на порушення доступності сервісу, отримання несанкціонованого доступу, викрадення даних, зміну інформації, перевірку вразливостей або подальше проникнення в інфраструктуру організації. Саме тому аналіз журналів подій вебсервера має розглядатися не лише як технічний інструмент адміністрування, а як важливий елемент системи кібербезпеки, що дозволяє виявляти потенційні ознаки атак ще на ранніх етапах. У сучасних підходах до безпеки вебзастосунків перелік основних ризиків часто пов’язується з класифікацією OWASP Top 10, яка визначає найбільш критичні напрями загроз для вебсистем, зокрема порушення контролю доступу, ін’єкції, помилки конфігурації, проблеми автентифікації, недоліки моніторингу та серверні запити до сторонніх ресурсів [11]. OWASP прямо визначає Top 10 як стандартний документ обізнаності для розробників і фахівців із безпеки вебзастосунків, що відображає широкий консенсус щодо критичних ризиків вебсистем.

Одним із найбільш поширених видів атак на вебсервери та вебзастосунки є ін’єкційні атаки, серед яких особливе місце посідає SQL-ін’єкція. Її сутність полягає у спробі передати до вебзастосунку спеціально сформовані дані, які можуть бути помилково інтерпретовані серверною логікою як частина запиту до бази даних. Унаслідок цього зловмисник теоретично може отримати доступ до даних, змінити їх, обійти механізми автентифікації або спричинити некоректну роботу системи. У журналах подій вебсервера ознаки SQL-ін’єкцій можуть проявлятися через підозрілі параметри в URL, нетипові символи у рядках запитів, повторювані звернення до сторінок пошуку, авторизації або фільтрації даних, збільшення кількості помилок 400, 403 або 500, а також через часті запити до тих

самих ресурсів із різними варіантами параметрів. Українські дослідники К. Терещенко, Т. Терещенко, Ю. Черниш, Р. Штонда та О. Бокій підкреслюють, що SQL-ін’єкції залишаються актуальною проблемою для вебзастосунків, оскільки пов’язані з обробкою даних, які надходять від користувача, та можуть створювати ризики несанкціонованого доступу до баз даних [12]. У цій частині важливо враховувати, що сам факт наявності підозрілого символу в URL не завжди доводить атаку, однак сукупність таких ознак у часовій послідовності формує важливий індикатор ризику.

Іншим поширеним видом атак є міжсайтовий скриптинг, який зазвичай позначається як XSS. Така атака пов’язана зі спробою впровадження шкідливого сценарію у вебсторінку або параметр запиту, якщо вебзастосунок некоректно обробляє введені користувачем дані. У журналах вебсервера XSS-активність може проявлятися через наявність у URL або тілі запиту підозрілих фрагментів, пов’язаних зі скриптами, обробниками подій, HTML-тегами, нетиповим кодуванням символів або багаторазовими зверненнями до форм введення. Оскільки вебсерверний лог найчастіше фіксує саме рядок HTTP-запиту, він може не відображати повний зміст тіла POST-запиту, якщо така деталізація не налаштована окремо. Тому для виявлення XSS важливо аналізувати не лише стандартні access logs, а й прикладні журнали вебзастосунку, WAF-журнали та помилки, які виникають під час обробки введених даних. У дослідженні І. Тишика щодо механізмів обходу фаєрволів вебдодатків прямо розглядаються атаки типу SQL Injection, XSS та CSRF як поширені сценарії впливу на вебсистеми, що підтверджує їхню практичну значущість для аналізу безпеки вебінфраструктури [13]. У логах такі атаки не завжди виглядають очевидно, тому система аналізу повинна враховувати комбінацію ознак: зміст запиту, частоту повторень, коди відповіді, джерело активності та поведінку користувача після отримання відповіді сервера.

До важливих загроз для вебсерверів належать атаки, пов’язані з порушенням контролю доступу. Йдеться про ситуації, коли користувач або автоматизований клієнт намагається отримати доступ до ресурсу, який не повинен бути доступним

для нього за логікою системи. У журналах подій такі спроби можуть проявлятися через велику кількість запитів до адміністративних сторінок, закритих директорій, службових панелей, файлів конфігурації, резервних копій, сторінок керування користувачами або API-ендпоінтів, які не призначені для публічного використання. Характерними ознаками є повторювані відповіді 401, 403, 404, інколи 302 у разі перенаправлення на сторінку авторизації, а також звернення до типових шляхів на кшталт адміністративних URL, сторінок входу, панелей керування CMS або службових файлів. OWASP Top 10 визначає порушення контролю доступу як один із ключових ризиків вебзастосунків, оскільки саме через помилки у розмежуванні прав можуть виникати умови для несанкціонованого перегляду, зміни або видалення даних [14]. Для системи аналізу логів це означає, що варто оцінювати не лише сам факт помилки доступу, а й її повторюваність, послідовність, прив'язку до IP-адреси, User-Agent, часу та конкретного набору ресурсів.

Окрему групу становлять атаки на механізми автентифікації, зокрема спроби підбору паролів, словникові атаки, credential stuffing та багаторазові невдалі входи до облікових записів. У журналах вебсервера такі атаки можуть бути помітні через значну кількість POST-запитів до сторінки авторизації, часті відповіді 401, 403 або 200 із подальшим поверненням на сторінку входу, нетипову частоту запитів з однієї IP-адреси або, навпаки, розподілену активність із багатьох адрес. Якщо вебзастосунок веде окремий журнал автентифікації, у ньому можуть фіксуватися невдалі спроби входу, заблоковані облікові записи, зміни пароля або підозрілі сесії. Дослідження М. Маркевича, присвячене серверним атакам на реляційні та нереляційні бази даних, серед інших сценаріїв розглядає словникові та комбіновані атаки, а також наголошує на важливості обмеження кількості запитів, хешування паролів і розмежування доступу [14]. У контексті логів це особливо важливо, оскільки система аналізу повинна вміти розпізнавати не лише одиничні невдалі входи, які можуть бути звичайною помилкою користувача, а й системну повторювану активність, що свідчить про автоматизований підбір облікових даних.

Значну небезпеку для вебсерверів становлять атаки типу path traversal, пов’язані зі спробами звернення до файлів або директорій поза межами дозволеної області вебзастосунку. У логах такі спроби можуть проявлятися через нетипові послідовності переходу між директоріями, закодовані символи, звернення до системних файлів, конфігураційних ресурсів, резервних копій або службових директорій. На практиці такі запити часто завершуються кодами 400, 403 або 404, однак уразливий застосунок може помилково обробити їх і надати доступ до небажаного ресурсу. Для системи аналізу логів важливо фіксувати не лише конкретні заборонені шаблони у URL, а й загальну поведінку клієнта: послідовне звернення до великої кількості потенційно чутливих шляхів, повторні спроби з різними варіантами кодування, зміну HTTP-методів і чергування помилкових запитів з успішними відповідями. MITRE CWE розглядає path traversal як клас програмних помилок, пов’язаних із некоректним обмеженням імені шляху до директорії, що підтверджує його значення для аналізу веббезпеки [15]. Для дипломної роботи це має прикладне значення, оскільки в алгоритмі оцінювання ризику можна передбачити підвищення балу небезпеки для запитів, які містять ознаки виходу за межі дозволеної структури каталогів.

До поширених сценаріїв атак належить сканування вебресурсу, яке може не бути атакою в завершеному вигляді, але часто передує їй. Автоматизовані сканери перевіряють наявність відомих директорій, адміністративних сторінок, резервних файлів, старих версій CMS, публічних API, тестових сторінок, файлів конфігурації або типових вразливих компонентів. У журналах подій така активність проявляється через велику кількість запитів до різних URL за короткий проміжок часу, переважання відповідей 404 і 403, однаковий або підозрілий User-Agent, звернення до ресурсів, яких немає в навігаційній структурі сайту, а також відсутність звичайної поведінки користувача, наприклад переходів між сторінками, завантаження стилів чи зображень. На відміну від звичайного користувача, автоматизований сканер часто рухається не логікою сайту, а словником потенційних шляхів. У дослідженні А. Толкачової та А. Піскозуба, присвяченому методам тестування безпеки вебзастосунків, підкреслюється роль

динамічної оцінки захищеності та інструментів, здатних виявляти SQL-ін’єкції, XSS, CSRF, RCE та інші вразливості [16]. Для захисної системи аналізу логів важливо відрізняти легітимне тестування або пошукову індексацію від підозрілого сканування, що потребує врахування контексту, частоти, набору URL і характеру відповідей сервера.

Окрему категорію становлять атаки, спрямовані на порушення доступності вебсервера, зокрема DoS і DDoS-атаки прикладного рівня. На відміну від мережевих атак, які можуть бути помітні на рівні інфраструктури, HTTP flood спрямований на перевантаження вебсервера великою кількістю HTTP-запитів. У логах така атака може проявлятися через різке зростання кількості однотипних запитів, надмірну частоту звернень до ресурсоємних сторінок, повторювані GET або POST-запити, збільшення кількості помилок 429, 500, 502, 503 або 504, а також через аномальне навантаження з однієї або багатьох IP-адрес. Cloudflare характеризує HTTP flood як DDoS-атаку, що намагається перевантажити цільовий сервер HTTP-запитами, а серед засобів протидії називає WAF, репутаційні бази IP та оперативний аналіз трафіку [17]. Для системи аналізу логів це означає необхідність побудови часових метрик: кількість запитів на хвилину, кількість унікальних IP-адрес, частка помилкових відповідей, частота звернень до конкретного ресурсу, середній обсяг відповіді та динаміка навантаження.

Важливим різновидом атак є експлуатація помилок конфігурації вебсервера або вебзастосунку. Йдеться про доступні службові сторінки, відкриті директорії, тестові файли, резервні копії, старі версії компонентів, невимкнені debug-режими, надмірно інформативні повідомлення про помилки, некоректні права доступу або небезпечні HTTP-заголовки. У журналах подій такі проблеми можуть проявлятися через звернення до конфігураційних файлів, неочікувані успішні відповіді на запити до службових ресурсів, велику кількість помилок 500 після передачі нетипових параметрів, а також повторювані спроби визначити технологічний стек застосунку. OWASP Top 10 окремо виділяє security misconfiguration як критичний ризик вебзастосунків, що підтверджує значення цього напряму для аналізу логів [18]. У практичному сенсі система аналізу повинна не тільки шукати явні ознаки

атак, а й допомагати адміністратору виявляти слабкі місця конфігурації через аналіз того, які ресурси запитуються, які відповіді повертаються та чи не демонструє сервер зайву технічну інформацію.

До атак на вебсервери також належать спроби використання вразливих або застарілих компонентів. Багато вебресурсів працюють на основі CMS, фреймворків, бібліотек, плагінів і додаткових модулів, які можуть мати відомі вразливості. У логах такі спроби часто проявляються через звернення до типових шляхів конкретних CMS, перевірку файлів версій, плагінів, адміністративних сторінок, API-ендпоінтів або відомих службових скриптів. Якщо вебзастосунок не використовує відповідну технологію, більшість таких запитів завершиться кодом 404, однак сама їхня поява свідчить про автоматизоване сканування. Якщо ж частина запитів завершується кодом 200 або 302, це може потребувати додаткової перевірки, оскільки сервер підтверджує наявність певного ресурсу. У межах аналізу логів варто виділяти звернення до шаблонних шляхів CMS, файлів резервних копій, архівів, інсталяційних директорій і старих службових сторінок. Такі ознаки не завжди свідчать про успішну атаку, але вони допомагають сформувати карту потенційних ризиків і виявити компоненти, які можуть потребувати оновлення або закриття доступу.

Потенційно небезпечними є також атаки, пов’язані з SSRF, тобто спробами змусити сервер виконати запит до внутрішнього або зовнішнього ресурсу від імені самого сервера. На рівні звичайного access log така атака може бути менш помітною, оскільки ключова дія відбувається всередині серверної логіки або у вихідному трафіку [19]. Однак певні ознаки можуть бути зафіксовані у вхідних запитах: передача URL як параметра, звернення до функцій завантаження зовнішніх ресурсів, незвичні параметри callback, webhook, image, file, redirect, а також помилки, що виникають під час спроби серверної обробки таких адрес. OWASP Top 10 виділяє SSRF як окрему категорію ризику, що свідчить про його значення для сучасних вебсистем [20]. Для повноцінного виявлення SSRF бажано поєднувати аналіз вебсерверних логів із прикладними журналами, логами вихідних мережевих з’єднань, WAF-подіями та журналами помилок. У дипломній

системі можна закласти базовий рівень виявлення через аналіз параметрів URL і помилок, однак у висновках доцільно зазначити, що більш точне виявлення SSRF потребує розширеного моніторингу серверної взаємодії.

Окрему увагу слід приділити CSRF-атакам, які пов’язані з небажаним виконанням дії від імені автентифікованого користувача. На відміну від SQL-ін’єкції або XSS, CSRF складніше виявити лише за стандартними access logs, оскільки запит може виглядати як легітимний і надходити від реального користувача. Проте певні непрямі ознаки можуть бути помітні: незвичні POST-запити без типового попереднього переходу, відсутність очікуваного referer, нетипова послідовність дій, виконання важливої операції одразу після входу, а також звернення до функцій зміни даних без характерної поведінки користувача в інтерфейсі. У дослідженні І. Тишика CSRF розглядається разом із SQL Injection та XSS у контексті перевірки вебзастосунків і механізмів WAF, що підтверджує доцільність урахування цієї загрози під час аналізу подій вебсервера [11]. У системі аналізу логів CSRF доцільно розглядати не як окремий очевидний шаблон, а як поведінкову аномалію, яка потребує кореляції між методом запиту, адресою ресурсу, наявністю referer, сесією користувача, часовою послідовністю та критичністю виконаної дії.

Ознаки атак у журналах подій доцільно групувати за кількома основними категоріями. До змістових ознак належать підозрілі параметри URL, нетипові символи, спроби звернення до службових файлів, ознаки ін’єкцій, скриптових фрагментів або переходу за межі дозволених директорій. До поведінкових ознак належать висока частота запитів, повторюваність однотипних дій, сканування великої кількості шляхів, нетипова послідовність переходів, різке збільшення помилок або активність у незвичний час. До технічних ознак можна віднести аномальні HTTP-методи, підозрілі User-Agent, відсутні або нетипові заголовки, коди відповідей 401, 403, 404, 405, 429, 500, 503, а також різке зростання часу обробки запитів або обсягу переданих даних. Для практичної реалізації системи аналізу логів важливо не обмежуватися одним правилом, оскільки реальна підозріла активність зазвичай проявляється через сукупність ознак. Саме тому

доцільним є використання системи ризикових балів, у якій кожна ознака підвищує або знижує загальну оцінку події.

Важливо враховувати, що не кожний підозрілий запис у журналі подій є доказом реальної атаки. У логах можуть фіксуватися помилки звичайних користувачів, запити пошукових роботів, технічні перевірки, звернення моніторингових сервісів, некоректні URL через помилки в інтерфейсі, старі посилання або тестування, яке виконується адміністратором. Тому система аналізу повинна передбачати ймовірнісний підхід до оцінювання ризику, а не автоматичне ототожнення будь-якої аномалії з кібератакою. У цьому контексті доцільно поєднувати сигнатурний підхід, який шукає відомі шаблони атак, із поведінковим аналізом, що виявляє відхилення від нормальної активності. Також важливо вести журнал спрацювань самої системи аналізу, щоб адміністратор міг бачити, чому конкретна подія отримала підвищений рівень ризику, які ознаки були виявлені та чи потрібно перевіряти цей випадок вручну.

Для теми розробки системи аналізу логів вебсервера особливе значення має зв’язок між типом атаки та конкретними полями логів. SQL-ін’єкції, XSS і path traversal найчастіше проявляються в полі URL, параметрах запиту, кодах помилок і повторюваності звернень. Brute force та credential stuffing помітні через часті запити до сторінки авторизації, невдалі відповіді та повторювану активність за короткий час. Сканування вразливостей фіксується через велику кількість різних URL, переважання помилок 404 і звернення до типових службових шляхів. HTTP flood виявляється через часову концентрацію запитів, зростання навантаження та збільшення помилок доступності. Помилки конфігурації проявляються через успішні відповіді на запити до ресурсів, які не повинні бути відкритими. SSRF і CSRF складніше виявляються лише за access logs, але можуть бути частково визначені через аналіз параметрів, referer, POST-запитів і прикладних журналів.

Отже, основні види кібератак на вебсервери мають різні технічні механізми, однак більшість із них залишає певні сліди в журналах подій. Саме ці сліди дають змогу побудувати систему первинного виявлення потенційних загроз, яка не замінює повноцінний комплекс кіберзахисту, але значно підвищує здатність

адміністратора своєчасно помічати підозрілу активність. Найбільш значущими для аналізу є такі параметри, як IP-адреса, час запиту, HTTP-метод, URL, код відповіді, User-Agent, referer, частота звернень і послідовність дій. Сукупний аналіз цих даних дозволяє виявляти ознаки SQL-ін’єкцій, XSS, path traversal, brute force, сканування, HTTP flood, помилок конфігурації, атак на застарілі компоненти, SSRF і CSRF. Саме тому журнали вебсервера слід розглядати як важливу інформаційну основу для розробки системи моніторингу, оцінювання ризику та виявлення потенційних кібератак.

### 1.3. Методи аналізу серверних логів для виявлення підозрілої активності

Аналіз серверних логів є одним із ключових напрямів технічного моніторингу вебінфраструктури, оскільки саме журнали подій дозволяють перейти від загального уявлення про функціонування вебсервера до конкретного виявлення ознак підозрілої активності. Якщо в попередніх підрозділах логи розглядалися як джерело інформації про запити, помилки, коди відповідей, IP-адреси, User-Agent та інші параметри, то в межах цього підрозділу основна увага зосереджується на методах їхньої обробки, інтерпретації та використання для виявлення потенційних кіберзагроз. Практична складність такого аналізу полягає в тому, що серверні журнали зазвичай мають великий обсяг, можуть містити технічний шум, повторювані записи, запити до статичних ресурсів, звернення пошукових роботів, помилки звичайних користувачів та окремі аномальні події, які необхідно відрізнити від реальних ознак атаки. Тому ефективна система аналізу логів повинна поєднувати методи попередньої обробки даних, структуризації записів, фільтрації, статистичного аналізу, сигнатурного виявлення, поведінкового аналізу, кореляції подій, візуалізації результатів і, за можливості, застосування елементів машинного навчання [21].

Початковим етапом аналізу серверних логів є парсинг, тобто перетворення текстових записів журналу у структуровану форму, придатну для подальшої автоматизованої обробки. У класичних access logs Apache або Nginx кожний запис

містить набір полів, однак у файлі журналу він зазвичай зберігається як один текстовий рядок. Для системи аналізу необхідно виділити з нього IP-адресу, часову мітку, HTTP-метод, URL, версію протоколу, код відповіді, розмір відповіді, referer, User-Agent та інші дані залежно від формату журналювання. Документація Apache підкреслює гнучкість логування через директиву LogFormat, яка дозволяє визначати склад записів журналу, а документація Nginx також передбачає налаштування форматів access log для збереження потрібних параметрів запиту [22]. Для дипломної системи це означає, що метод парсингу повинен бути достатньо гнучким, адже в реальних умовах формат логів може відрізнятися залежно від конфігурації сервера, використання проксі, CDN, балансувальника навантаження або додаткових модулів безпеки.

Після парсингу важливим етапом є нормалізація даних. Вона полягає у приведенні різних полів логів до єдиного формату, що забезпечує можливість їх подальшого порівняння, групування й аналізу. Часові мітки можуть переводитися до єдиного часового поясу, URL можуть декодуватися та очищуватися від зайвих символів, HTTP-методи переводитися до єдиного регістру, коди відповідей групуватися за класами 2xx, 3xx, 4xx, 5xx, а IP-адреси можуть агрегуватися для підрахунку кількості запитів. Нормалізація є особливо важливою для виявлення атак, які використовують кодування символів або варіативність запису параметрів. Наприклад, підозрілий запит може бути представлений у різних формах через URL encoding, подвійне кодування або зміну регістру символів. Якщо система аналізу не нормалізує такі значення, вона може не розпізнати однакові за сутністю запити як пов’язані між собою. Саме тому якісна попередня обробка логів є основою для подальшого сигнатурного, статистичного та поведінкового аналізу.

Одним із найпростіших і водночас практично корисних методів виявлення підозрілої активності є фільтрація та групування записів за ключовими параметрами. На цьому етапі система може відокремлювати запити до статичних ресурсів, таких як зображення, CSS-файли, JavaScript-файли або іконки, якщо вони не мають значення для конкретного сценарію безпекового аналізу. Після цього записи можуть групуватися за IP-адресою, часовим інтервалом,

HTTP-методом, кодом відповіді, URL або User-Agent. Такий підхід дозволяє швидко виявити адреси, з яких надходить найбільше запитів, ресурси, що найчастіше викликають помилки, періоди пікового навантаження, а також повторювані звернення до адміністративних або неіснуючих сторінок. У практичному плані це дає змогу побачити не окремий запис журналу, а поведінкову картину активності. Саме групування є базою для подальшого обчислення показників ризику, оскільки одиничний запит до неіснуючої сторінки може бути випадковим, тоді як десятки або сотні таких запитів за короткий час з однієї адреси можуть свідчити про сканування вебресурсу.

Сигнатурний метод аналізу логів базується на пошуку заздалегідь визначених шаблонів, які характерні для відомих типів атак. До таких шаблонів можуть належати фрагменти URL, параметри запиту, підозрілі символи, типові назви службових файлів, спроби звернення до конфігураційних ресурсів, ознаки SQL-ін’єкцій, XSS, path traversal або автоматизованого сканування. Перевагою сигнатурного підходу є його зрозумілість, простота реалізації та можливість пояснити, чому конкретний запис отримав статус підозрілого. Недоліком є залежність від наперед визначених правил, через що система може не виявляти нові або модифіковані сценарії атак. У дослідженнях щодо методів виявлення шкідливого програмного забезпечення сигнатурний аналіз розглядається як традиційний підхід, який є ефективним для відомих загроз, але потребує доповнення евристичними та інтелектуальними методами [23]. Для аналізу вебсерверних логів така логіка також є актуальною: сигнатури добре працюють для типових шаблонів, але не можуть бути єдиним механізмом виявлення підозрілої активності.

Евристичний метод аналізу доповнює сигнатурний підхід і ґрунтується не лише на пошуку конкретного шаблону, а й на оцінці загальної підозрілості дії. Наприклад, запит до сторінки авторизації не є небезпечним сам по собі, однак велика кількість таких запитів за короткий проміжок часу може бути ознакою підбору облікових даних. Так само відповідь 404 не є атакою, але масове звернення до великої кількості неіснуючих URL може вказувати на сканування.

Евристичні правила можуть враховувати частоту запитів, кількість помилок, нетипові HTTP-методи, звернення до заборонених шляхів, підозрілий User-Agent, повторюваність параметрів, різке збільшення навантаження або нетипову послідовність дій. У системі аналізу логів евристичний підхід зручно реалізовувати через механізм балів ризику, коли кожна виявлена ознака додає певну кількість балів до загальної оцінки події або IP-адреси. Наприклад, підозрілий URL може додати один рівень ризику, часті помилки 403 — інший, а надмірна кількість запитів за короткий час — ще один. У результаті адміністратор отримує не просто список усіх записів, а ранжований перелік подій за ступенем потенційної небезпеки.

Статистичний аналіз серверних логів передбачає вивчення кількісних закономірностей активності вебресурсу. До таких показників належать загальна кількість запитів за певний період, кількість унікальних IP-адрес, середня частота звернень, частка помилок 4xx і 5xx, кількість запитів до конкретних URL, кількість невдалих спроб доступу, середній розмір відповіді, співвідношення GET і POST-запитів, кількість запитів із порожнім або підозрілим User-Agent. Статистичний метод дозволяє визначити базову норму активності, з якою надалі порівнюються нові події. Якщо у звичайних умовах вебресурс отримує певну середню кількість запитів на хвилину, то різке перевищення цього рівня може бути ознакою навантажувальної атаки або масового сканування. Якщо частка помилок 404 раптово зростає, це може свідчити про перевірку неіснуючих шляхів. Якщо різко збільшується кількість POST-запитів до сторінки входу, це може вказувати на атаку на автентифікацію. У контексті систем підтримки прийняття рішень українські дослідники наголошують, що автоматизація аналізу великих потоків даних є важливою передумовою ефективного моделювання систем захисту інформації [24].

Поведінковий аналіз є більш складним методом, оскільки він орієнтується не лише на окремі параметри запиту, а й на послідовність дій користувача або групи користувачів. Звичайний відвідувач вебсайту зазвичай рухається певною логікою: відкриває головну сторінку, переходить за внутрішніми посиланнями,

завантажує статичні ресурси, виконує пошук, авторизується або переглядає контент. Автоматизований сканер або бот часто діє інакше: надсилає багато запитів до непов’язаних сторінок, не завантажує допоміжні ресурси, швидко змінює URL, перевіряє типові адміністративні шляхи, виконує запити з однаковими інтервалами або використовує підозрілий User-Agent. Поведінковий аналіз дозволяє виявляти такі відмінності, навіть якщо конкретний запит не містить очевидної сигнатури атаки. Практично це може реалізовуватися через побудову профілю активності IP-адреси або сесії: кількість URL, кількість помилок, часові інтервали, послідовність переходів, співвідношення успішних і помилкових відповідей, повторюваність дій і відхилення від типової поведінки.

Кореляційний аналіз подій є важливим для ситуацій, коли одна подія не має достатнього значення, але сукупність кількох подій формує ознаку потенційної атаки. Наприклад, спочатку з однієї IP-адреси можуть надходити запити до неіснуючих сторінок, потім звернення до адміністративної панелі, далі кілька невдалих спроб авторизації, а після цього запит із підозрілим параметром у URL. Кожна з цих подій окремо може бути не критичною, однак їхня послідовність свідчить про підозрілу активність. Саме тому в системах моніторингу безпеки використовуються підходи до кореляції подій, які дозволяють поєднувати дані з різних джерел: вебсерверних логів, журналів автентифікації, WAF, IDS/IPS, баз даних, системних журналів і мережевих сенсорів. OWASP Logging Cheat Sheet рекомендує розглядати логування як частину централізованої інфраструктури моніторингу та аналізу, що має підтримувати виявлення подій безпеки, розслідування інцидентів і реагування [25]. Для дипломної системи базовий варіант кореляції може полягати у зв’язуванні подій за IP-адресою, часовим проміжком, URL, кодом відповіді та типом виявленої ознаки.

Окремим напрямом є використання індикаторів компрометації та списків відомих небезпечних ознак. Індикатори компрометації можуть включати підозрілі IP-адреси, домени, User-Agent, хеші файлів, URL-шаблони, характерні параметри запитів або інші ознаки, які вже пов’язані з відомою шкідливою активністю. У вебсерверних логах такі індикатори можуть використовуватися для швидкої

фільтрації подій і підвищення рівня ризику для запитів, які збігаються з відомими небезпечними ознаками. Водночас застосування індикаторів не повинно бути механічним, оскільки IP-адреси можуть змінювати власників, User-Agent легко підробити, а частина шаблонів може давати хибні спрацювання. Українське дослідження щодо використання індикаторів компрометації для виявлення кібератак розглядає IDS, IPS, Splunk, методи машинного навчання та штучного інтелекту як компоненти сучасних рішень у сфері кібербезпеки [26]. Це підтверджує доцільність поєднання локального аналізу логів із ширшим контекстом подій безпеки та зовнішніми джерелами інформації про загрози.

Методи машинного навчання в аналізі логів використовуються для автоматизованого виявлення аномалій, класифікації подій і прогнозування ризиків. На відміну від сигнатурного підходу, який шукає наперед відомі шаблони, машинне навчання може виявляти нетипові комбінації ознак, які не були явно задані розробником системи. Для аналізу серверних логів можуть застосовуватися алгоритми кластеризації, дерева рішень, Isolation Forest, методи класифікації, нейронні мережі, послідовні моделі та інші підходи. Наприклад, Isolation Forest може використовуватися для виявлення записів або IP-адрес, поведінка яких суттєво відрізняється від більшості інших. У дослідженні щодо безпеки корпоративних баз даних із використанням SIEM та AlienVault описано застосування алгоритму Isolation Forest для моделі виявлення аномалій доступу на основі історичних даних [27]. Хоча така робота стосується баз даних, її підхід може бути адаптований до вебсерверних логів, оскільки в обох випадках ідеться про аналіз подій доступу, пошук аномальних дій і оцінювання потенційного ризику.

Разом із тим використання машинного навчання в дипломній системі потребує обережності. Для якісної роботи моделі потрібні достатні обсяги навчальних даних, коректне маркування подій, очищення набору даних, вибір ознак, тестування точності та аналіз хибних спрацювань. Якщо даних мало або вони не відображають реальну різноманітність атак і нормальної поведінки, модель може працювати нестабільно. Сучасні дослідження log-based anomaly

detection показують, що аналіз журналів часто використовує кількісні, послідовні та глибинні моделі, однак ефективність таких підходів залежить від структури даних, якості ознак і умов застосування [28]. Тому в межах практичної роботи доцільно передбачити базову rule-based або risk-score модель як основну, а машинне навчання розглядати як перспективний напрям подальшого розвитку системи. Такий підхід буде обґрунтованим, оскільки він дозволить отримати прозору логіку роботи системи, яку легко пояснити, протестувати й адаптувати.

Важливим методом є часовий аналіз, який дозволяє виявляти підозрілу активність через інтенсивність і ритм запитів. Для цього записи логів групуються за інтервалами: секундами, хвилинами, годинами або днями. Після цього система може визначати кількість запитів у кожному інтервалі, кількість унікальних IP-адрес, кількість помилок, частоту звернень до певного URL, кількість POST-запитів до сторінки входу або кількість відповідей 5xx. Часовий аналіз особливо корисний для виявлення HTTP flood, brute force, сканування директорій і раптових піків помилок. Якщо система бачить, що за одну хвилину з однієї IP-адреси надійшло значно більше запитів, ніж у середньому, вона може підвищити рівень ризику [29]. Якщо протягом короткого часу багато різних IP-адрес звертаються до одного ресурсоємного URL, це може бути ознакою розподіленої атаки або бот-активності. Такий метод не вимагає складних моделей, але дає практичний результат для первинного моніторингу.

Візуалізація результатів є не лише допоміжним, а й важливим аналітичним методом, оскільки адміністратор або фахівець із безпеки повинен швидко бачити загальну картину подій. Табличне представлення логів зручно для детального перегляду окремих записів, однак графіки, діаграми та інформаційні панелі дозволяють краще виявляти тенденції. Доцільно візуалізувати кількість запитів за часом, розподіл кодів відповідей, найактивніші IP-адреси, найбільш проблемні URL, кількість подій за рівнями ризику, частку підозрілих запитів і динаміку помилок. Візуалізація допомагає швидко помітити піки навантаження, зростання кількості помилок, появу нетипових джерел активності або зміну структури трафіку. У практичному значенні це підвищує зручність використання системи,

оскільки адміністратор отримує не лише сирі логи, а зрозумілий аналітичний інтерфейс для прийняття рішень.

Метод оцінювання ризику є центральним для розроблюваної системи, оскільки він дозволяє перетворити набір технічних ознак на узагальнений показник небезпеки. Такий метод може бути реалізований через систему правил і вагових коефіцієнтів. Наприклад, запит до адміністративної сторінки може отримувати один бал ризику, підозрілий параметр у URL — два бали, часті помилки 403 або 404 — додаткові бали, велика кількість запитів за короткий час — ще вищий рівень ризику. Після підсумовування система може класифікувати події як низькоризикові, середньоризикові або високоризикові. Перевага такого методу полягає в прозорості: адміністратор може бачити, які саме ознаки вплинули на результат. Крім того, ваги можна змінювати залежно від особливостей конкретного вебресурсу. Наприклад, для сайту з відкритою API-інфраструктурою велика кількість запитів може бути нормальною, тоді як для невеликого інформаційного сайту така активність може бути підозрілою.

У практиці безпекового моніторингу значення має також метод зменшення хибних спрацювань. Надмірна кількість попереджень може знизити ефективність системи, оскільки адміністратор перестає реагувати на сигнали, які часто не підтверджуються. Для зменшення хибних спрацювань доцільно використовувати whitelist для відомих службових IP-адрес, відокремлювати легітимних пошукових роботів, враховувати звичайні піки активності, не підвищувати рівень ризику лише за однією слабкою ознакою, а також зберігати пояснення для кожного спрацювання. Важливим є й те, що підозріла активність не завжди означає успішну атаку [30]. Система аналізу логів має допомагати виявляти потенційні інциденти, але остаточне рішення щодо реагування може потребувати додаткової перевірки. Саме тому в інтерфейсі доцільно передбачити фільтри, сортування, перегляд деталей події та можливість аналізу історії активності конкретної IP-адреси.

Узагальнюючи, можна зазначити, що методи аналізу серверних логів для виявлення підозрілої активності мають комплексний характер і повинні

застосовуватися у взаємозв’язку. Парсинг і нормалізація забезпечують підготовку даних, фільтрація та групування дозволяють зменшити обсяг технічного шуму, сигнатурний аналіз виявляє відомі шаблони атак, евристичні правила допомагають оцінювати підозрілість дій, статистичний і часовий аналіз дозволяють бачити аномальну інтенсивність, поведінковий підхід виявляє нетипові послідовності, кореляція подій формує ширший контекст, а візуалізація робить результати зрозумілими для адміністратора. Для розроблюваної системи найбільш доцільним є поєднання rule-based підходу, статистичного аналізу та оцінювання ризику, оскільки така модель є прозорою, придатною для реалізації в межах дипломної роботи та достатньою для виявлення базових ознак потенційних кібератак. У перспективі така система може бути розширена за рахунок інтеграції з SIEM, WAF, зовнішніми індикаторами компрометації, а також модулями машинного навчання для більш гнучкого виявлення аномалій.

**Висновки до розділу 1**

У першому розділі було розглянуто теоретичні основи аналізу логів вебсервера в системі кібербезпеки та визначено їхнє значення для виявлення потенційно підозрілої активності. Встановлено, що логи вебсервера є важливим джерелом технічної та безпекової інформації, оскільки вони фіксують фактичні звернення користувачів, автоматизованих клієнтів, ботів і потенційних зловмисників до вебресурсу. На основі журналів подій можна отримати дані про IP-адресу джерела запиту, часову мітку, HTTP-метод, URL-адресу, код відповіді сервера, обсяг переданих даних, User-Agent та інші параметри, які мають значення для аналізу поведінки клієнтів і виявлення ознак кібератак.

З’ясовано, що структура логів вебсервера має прикладне значення для побудови системи моніторингу, оскільки кожне поле запису може використовуватися як окрема аналітична ознака. IP-адреса дозволяє групувати активність за джерелами звернень, часова мітка дає можливість оцінювати інтенсивність запитів, HTTP-метод відображає характер взаємодії з ресурсом, URL вказує на об’єкт звернення, а код відповіді сервера допомагає визначити

результат обробки запиту. При цьому встановлено, що жоден із цих параметрів не є самодостатнім доказом атаки, тому ефективне виявлення підозрілої активності потребує комплексного аналізу кількох ознак у їх взаємозв’язку.

У межах розділу було охарактеризовано основні види кібератак на вебсервери та визначено їхні типові прояви в журналах подій. До таких атак належать SQL-ін’єкції, XSS-атаки, спроби порушення контролю доступу, brute force та credential stuffing, path traversal, автоматизоване сканування директорій, HTTP flood, атаки на застарілі компоненти, експлуатація помилок конфігурації, а також складніші сценарії на зразок SSRF і CSRF. Показано, що більшість таких загроз залишає певні непрямі або прямі сліди в логах вебсервера, зокрема підозрілі параметри URL, часті звернення до адміністративних сторінок, велику кількість відповідей 401, 403 або 404, нетипові HTTP-методи, різке зростання кількості запитів, підозрілі User-Agent або повторювану активність з однієї чи кількох IP-адрес.

Також було встановлено, що аналіз серверних логів має спиратися не лише на пошук окремих підозрілих записів, а й на виявлення поведінкових закономірностей. Одинична помилка доступу або один запит до неіснуючої сторінки можуть бути звичайною дією користувача, однак їх масове повторення протягом короткого проміжку часу може свідчити про сканування, підбір облікових даних або іншу небезпечну активність. Саме тому важливими є часовий аналіз, групування подій за IP-адресами, оцінювання частоти звернень, аналіз послідовності дій, визначення частки помилкових відповідей та зіставлення декількох параметрів журналу в межах одного сценарію.

У розділі було розглянуто основні методи аналізу серверних логів, серед яких парсинг, нормалізація, фільтрація, групування, сигнатурний аналіз, евристичний підхід, статистичний аналіз, поведінковий аналіз, кореляція подій, використання індикаторів компрометації, візуалізація та елементи машинного навчання. Визначено, що для практичної реалізації системи аналізу логів у межах дипломної роботи найбільш доцільним є поєднання rule-based підходу, статистичних показників і механізму оцінювання рівня ризику. Така модель є

достатньо зрозумілою, прозорою для пояснення результатів, придатною для тестування та може бути розширена в майбутньому за рахунок інтеграції з SIEM, WAF, зовнішніми базами індикаторів компрометації або модулями машинного навчання.

Отже, результати першого розділу дали змогу сформувати теоретичну основу для подальшого проєктування системи аналізу логів вебсервера. Було доведено, що журнали подій є не лише технічним інструментом адміністрування, а й важливим джерелом даних для кібербезпеки, оскільки дозволяють виявляти потенційні ознаки атак, аналізувати поведінку користувачів, оцінювати рівень ризику та підтримувати прийняття рішень адміністратором. Саме ці положення є підґрунтям для наступного розділу, у якому буде розглянуто проєктування системи аналізу логів вебсервера, визначення її функціональних вимог, алгоритму обробки даних та архітектури програмного рішення.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ АНАЛІЗУ ЛОГІВ ВЕБСЕРВЕРА ДЛЯ ВИЯВЛЕННЯ ПОТЕНЦІЙНИХ КІБЕРАТАК

## 2.1. Формування функціональних вимог до системи аналізу логів

Формування функціональних вимог до системи аналізу логів вебсервера є важливим етапом проєктування, оскільки саме на цьому рівні визначається, які задачі повинна виконувати система, які дані вона має обробляти, які результати повинна надавати користувачу та яким чином має підтримувати процес виявлення потенційних кібератак. Якщо теоретичний розділ роботи був спрямований на з'ясування сутності логів, їхньої структури, основних типів атак і методів аналізу, то другий розділ має прикладний характер і безпосередньо пов'язаний із побудовою програмного рішення. У цьому контексті функціональні вимоги виступають основою для подальшого розроблення алгоритму обробки логів, проєктування архітектури системи, структури бази даних, адміністративної панелі та механізмів візуалізації результатів.

Функціональні вимоги до системи аналізу логів вебсервера мають формуватися з урахуванням особливостей самих журналів подій, типових сценаріїв роботи адміністратора, можливих ознак кібератак і практичних обмежень дипломного проєкту. Важливо, щоб система не була надмірно складною, однак водночас виконувала ключові задачі безпекового моніторингу: приймала або завантажувала лог-файл, коректно розпізнавала записи, виділяла основні поля, зберігала результати обробки, визначала підозрілі ознаки, обчислювала рівень ризику, відображала події у зручному інтерфейсі та давала можливість переглядати статистику. Такий підхід узгоджується з практикою безпечного проєктування вебсистем, де вимоги до моніторингу, журналювання та обробки подій мають закладатися ще на етапі створення програмного рішення, а не додаватися після виникнення інцидентів [31]. OWASP ASVS використовується як відкрита база вимог для перевірки безпеки вебзастосунків, зокрема містить окремий напрям, пов'язаний з обробкою помилок і логуванням.

Базовою функціональною вимогою є можливість завантаження логів вебсервера до системи. У межах дипломної роботи доцільно передбачити варіант

завантаження текстового файлу журналу через адміністративну панель або вибір локального файлу для подальшого аналізу. У реальних умовах така система може бути розширена за рахунок автоматичного зчитування файлів із директорії сервера, підключення до syslog, отримання логів із контейнерів, хмарних сервісів або систем централізованого моніторингу. Однак для навчального програмного рішення достатньо реалізувати завантаження файлу у форматі access log, після чого система повинна перевірити його на наявність записів, базову коректність структури та можливість подальшого парсингу. Оскільки Apache і Nginx підтримують налаштування формату журналювання через відповідні директиви LogFormat і log_format, система повинна бути орієнтована щонайменше на один типовий формат, але з перспективою подальшого розширення [32]. Документація Apache вказує, що LogFormat визначає склад записів, а документація Nginx описує налаштування access_log і log_format для фіксації параметрів запиту.

Другою важливою вимогою є парсинг логів, тобто розбиття кожного рядка журналу на окремі структуровані поля. Система повинна виділяти IP-адресу, дату і час запиту, HTTP-метод, URL, версію протоколу, код відповіді, розмір відповіді, referer та User-Agent, якщо такі дані наявні у вихідному файлі. Саме ці поля є основою для подальшого аналізу, оскільки дозволяють групувати активність за джерелом, оцінювати інтенсивність запитів, визначати підозрілі URL, аналізувати помилки доступу та порівнювати поведінку різних клієнтів. На рівні програмної реалізації парсинг може здійснюватися за допомогою регулярного виразу, який відповідає обраному формату access log. Наприклад, для типового combined log format можна використати спрощений підхід такого вигляду:

```
import re
log_pattern = re.compile(
   r'(?P<ip>\S+) \S+ \S+ \[(?P<time>[^\]]+)\] '
   r'"(?P<method>\S+) (?P<url>\S+) (?P<protocol>[^"]+)" '
   r'(?P<status>\d{3}) (?P<size>\S+) '
   r'"(?P<referer>[^"]*)" "(?P<user_agent>[^"]*)"'
)
```

```
def parse_log_line(line):
    match = log_pattern.match(line)
    if not match:
        return None
    return match.groupdict()
```

Фрагмент коду демонструє загальну логіку роботи парсера: кожний рядок журналу перевіряється на відповідність шаблону, після чого перетворюється на словник із полями, придатними для подальшої обробки. У практичній системі цей механізм доцільно доповнити обробкою помилок, підтримкою різних форматів логів, перетворенням часу до стандартного формату та перевіркою типів даних. Це важливо тому, що реальні логи можуть містити пошкоджені рядки, нестандартні User-Agent, порожні поля, різні часові пояси або відмінності у форматі записів.

Функціональною вимогою є нормалізація отриманих даних. Після парсингу система повинна привести значення до уніфікованого вигляду: HTTP-методи до одного регістру, коди відповідей до числового типу, часові мітки до єдиного формату, URL до декодованої або очищеної форми, а порожні поля — до стандартизованого значення. Нормалізація необхідна для того, щоб однакові за змістом події не сприймалися системою як різні лише через відмінності у записі. Наприклад, підозрілі параметри URL можуть бути представлені в закодованому вигляді, а User-Agent може містити спеціальні символи або надто довгі рядки. Крім того, нормалізація зменшує ризик помилок під час фільтрації, групування та обчислення ризику. У контексті сучасного управління логами NIST SP 800-92 розглядає лог-менеджмент як процес, що включає генерацію, передавання, зберігання, аналіз і захист журналів подій, що підтверджує необхідність системної попередньої обробки логів перед їх використанням для безпекових висновків [33].

У серверних журналах значну частину записів можуть становити звернення до статичних файлів, зображень, стилів, JavaScript-файлів, favicon, шрифтів або інших допоміжних ресурсів. Не всі такі записи мають однакове значення для виявлення потенційних кібератак. Система повинна дозволяти або автоматично

відокремлювати такі записи, або надавати адміністратору можливість увімкнути фільтр для перегляду лише значущих подій. Водночас повне ігнорування статичних ресурсів не завжди є доцільним, оскільки в окремих випадках аномальна кількість запитів до великих файлів або ресурсів може свідчити про навантажувальну активність. Тому функціональна вимога має формулюватися гнучко: система повинна забезпечувати фільтрацію за типом ресурсу, але не повинна безумовно видаляти такі записи з бази даних. Це дозволить зберігати повноту даних і водночас робити інтерфейс зручнішим для аналітичного перегляду.

Важливим є виявлення підозрілих URL і параметрів запиту. Система повинна аналізувати поле URL на наявність типових ознак SQL-ін’єкцій, XSS, path traversal, звернення до службових файлів, спроб доступу до адміністративних сторінок, резервних копій, конфігураційних файлів або неіснуючих директорій. При цьому важливо, щоб механізм аналізу був не надто жорстким, адже окремі символи або слова можуть траплятися і в легітимних запитах. Найкращим варіантом для дипломної роботи є використання простого rule-based підходу, коли кожна підозріла ознака не блокує запит автоматично, а додає певну кількість балів до загального рівня ризику. Наприклад, звернення до /admin, /wp-login.php, /.env, /backup.zip або параметри з фрагментами script, union, select, ../ можуть бути підставою для позначення події як потенційно підозрілої. OWASP Logging Cheat Sheet наголошує, що безпекове логування має фіксувати події, корисні для виявлення інцидентів та подальшого розслідування, але не повинне перетворюватися на неконтрольоване накопичення зайвих або чутливих даних [34].

```
    SUSPICIOUS_PATTERNS = [
  "union", "select", "<script", "../", "%2e%2e",
  ".env", "wp-login", "admin", "backup", "phpinfo"
]
def detect_suspicious_url(url):
  normalized_url = url.lower()
```

```
    detected = []
    for pattern in SUSPICIOUS_PATTERNS:
        if pattern in normalized_url:
            detected.append(pattern)
    return detected
```

Наведений фрагмент коду демонструє базовий спосіб виявлення підозрілих URL за допомогою списку шаблонів. Такий підхід не є повноцінною системою захисту, однак він добре підходить для навчальної системи аналізу логів, оскільки дозволяє прозоро показати, чому конкретний запис був позначений як ризиковий. У подальшому цей механізм можна розширити шляхом використання регулярних виразів, окремих правил для різних типів атак, зовнішніх списків індикаторів компрометації або інтеграції з WAF-журналами.

Система повинна підраховувати кількість запитів від кожної IP-адреси за визначений часовий проміжок і порівнювати цей показник із допустимим порогом. Якщо одна адреса надсилає надто багато запитів за короткий час, це може свідчити про сканування, brute force або HTTP flood. Також система повинна враховувати кількість унікальних URL, кількість помилок 404, 403 або 401, кількість POST-запитів до сторінок авторизації та інші показники, які дають змогу виявити аномальну поведінку. OWASP Web Security Testing Guide зазначає, що аналіз логів може використовуватися не лише для статистики використання вебресурсу, а й для визначення того, чи відбуваються атаки на вебсервер або застосунок [35].

```
        from collections import defaultdict
from datetime import timedelta
def count_requests_by_ip(events, window_minutes=1):
    result = defaultdict(int)
    for event in events:
        key = event["ip"]
        result[key] += 1
    return dict(result)
```

```
def is_rate_suspicious(request_count, threshold=100):
    return request_count > threshold
```

Цей приклад є спрощеним, оскільки не враховує реальне ковзне часове вікно, але показує загальну логіку вимоги: система повинна не лише переглядати окремі записи, а й агрегувати події. У повнішій реалізації доцільно враховувати часові інтервали, наприклад кількість запитів від IP-адреси за 1 хвилину, 5 хвилин або 1 годину, а також будувати окремі показники для різних типів подій. Саме агрегування дозволяє відрізнити одиничну помилку користувача від системної підозрілої активності.

Вимогою є формування інтегрального рівня ризику для кожної події або групи подій. Такий рівень ризику може визначатися на основі суми балів, отриманих за різні ознаки: підозрілий URL, небезпечний HTTP-метод, код відповіді 401, 403 або 404, надмірна частота запитів, порожній або підозрілий User-Agent, спроба доступу до адміністративної сторінки, повторювані невдалі запити. Результатом має бути класифікація подій за рівнями, наприклад низький, середній і високий ризик. Така модель є доцільною для дипломної роботи, оскільки вона зрозуміла, прозора та легко пояснюється в тексті дослідження. Крім того, вона дозволяє адміністратору бачити причини спрацювання, а не лише отримувати абстрактне повідомлення про небезпеку.

```
    def calculate_risk(event, request_count=0):
    score = 0
    reasons = []
    if event["status"] in ["401", "403", "404"]:
        score += 1
        reasons.append("помилка доступу або неіснуючий ресурс")
    suspicious = detect_suspicious_url(event["url"])
    if suspicious:
        score += 2
        reasons.append("підозрілі шаблони в URL: " + ", ".join(suspicious))
    if request_count > 100:
```

```
        score += 3
        reasons.append("надмірна кількість запитів з IP-адреси")
    if score >= 5:
        level = "high"
    elif score >= 2:
        level = "medium"
    else:
        level = "low"
    return {"score": score, "level": level, "reasons": reasons}
```

Цей фрагмент коду ілюструє базову логіку risk-score моделі. Вона не претендує на абсолютну точність, однак є придатною для демонстрації принципу роботи системи: кожна ознака підозрілої активності впливає на загальну оцінку, а результат супроводжується поясненням. Саме пояснюваність є важливою вимогою до системи, оскільки адміністратор повинен розуміти, чому подія отримала певний рівень ризику. Без пояснення система перетворюється на непрозорий механізм, довіра до якого буде нижчою.

Після парсингу, нормалізації та оцінювання ризику система повинна зберігати записи у структурованому вигляді, щоб адміністратор міг повертатися до них, виконувати пошук, сортування, фільтрацію й аналіз за різними параметрами. Доцільно передбачити таблицю лог-подій, у якій зберігатимуться IP-адреса, час запиту, метод, URL, код відповіді, розмір відповіді, User-Agent, рівень ризику та пояснення спрацювання. Також можна передбачити окрему таблицю для збереження правил ризику або результатів агрегованого аналізу. Це створить основу для подальшої реалізації адміністративної панелі, статистичних віджетів і звітів.

Користувач системи, яким у межах роботи виступає адміністратор або фахівець із кібербезпеки, повинен мати можливість бачити загальний список подій, фільтрувати їх за рівнем ризику, IP-адресою, кодом відповіді, датою, методом або URL. Також доцільно передбачити окремий перегляд події, де буде показано вихідний запис логу, розпізнані поля, рівень ризику та причини, через

які система позначила подію як підозрілу. Важливо, щоб інтерфейс не перевантажував користувача зайвою інформацією, а допомагав швидко знаходити найбільш важливі події. Саме тому функціональна вимога до адміністративної панелі повинна передбачати сортування подій за рівнем ризику та часом, а також можливість швидкого перегляду високоризикових записів.

Система повинна формувати загальні метрики: кількість оброблених записів, кількість унікальних IP-адрес, кількість подій із низьким, середнім і високим ризиком, найактивніші IP-адреси, найчастіші коди відповідей, найбільш запитувані URL, кількість помилок 4xx і 5xx, а також динаміку запитів у часі. Візуалізація може бути реалізована у вигляді простих графіків, таблиць, діаграм або інформаційних карток. Вимога до візуалізації має практичне значення, оскільки адміністратору значно простіше помітити різке зростання кількості помилок або підозрілих запитів на графіку, ніж вручну переглядати сотні рядків журналу. У цьому аспекті система аналізу логів поєднує функції технічної обробки даних і підтримки прийняття рішень.

Якщо система позначає певну подію як підозрілу, вона повинна зберігати не лише результат, а й підстави такого рішення. Наприклад, якщо IP-адреса отримала високий рівень ризику через велику кількість запитів, наявність підозрілих параметрів і часті помилки 404, ці причини повинні бути доступні в інтерфейсі. Такий підхід підвищує прозорість системи та полегшує перевірку її роботи. Крім того, журнал спрацювань може використовуватися під час тестування системи на змодельованих сценаріях атак, оскільки дозволяє порівняти очікувані результати з фактичними.

Оскільки система працює з журналами вебсервера, у яких можуть міститися IP-адреси, URL, технічні параметри запитів і потенційно чутлива інформація, доступ до неї має бути обмежений. У межах дипломної роботи можна передбачити просту авторизацію адміністратора, захист сторінок адміністративної панелі, перевірку типу завантажуваного файлу, обмеження розміру файлу та недопущення виконання завантаженого вмісту як коду. Особливо важливо, щоб система не зберігала у відкритому вигляді зайві чутливі дані й не виводила

небезпечний вміст URL без екранування в HTML-інтерфейсі, оскільки самі логи можуть містити фрагменти XSS-атаки. Саме тому відображення даних із логів в адміністративній панелі повинно здійснюватися з урахуванням безпечного виводу.

Для перевірки працездатності потрібно мати тестові записи, які імітують нормальну активність, сканування директорій, спроби доступу до адміністративних сторінок, SQL-ін’єкції, XSS-параметри, надмірну кількість запитів і помилки доступу. Система повинна коректно обробляти такі записи та надавати очікуваний результат. Це дозволить у третьому розділі провести тестування розробленої системи на змодельованих сценаріях потенційних кібератак і оцінити її ефективність. NIST SP 800-137 розглядає безперервний моніторинг як процес підтримки обізнаності про стан інформаційної безпеки, що підтверджує важливість регулярної перевірки й аналізу подій, а не одноразового перегляду журналів [36].

Окремо варто визначити нефункціональні вимоги, хоча основний акцент у цьому підрозділі зроблено саме на функціональних можливостях. Система повинна бути зручною у використанні, працювати достатньо швидко для обробки середніх за обсягом лог-файлів, забезпечувати зрозумілу структуру інтерфейсу, не вимагати складного налаштування для базового запуску, бути придатною для подальшого розширення та мати зрозумілу логіку програмного коду. Також важливо, щоб система була модульною: окремо повинні існувати компоненти завантаження логів, парсингу, аналізу, обчислення ризику, збереження даних і відображення результатів. Такий підхід полегшує тестування, підтримку і подальше вдосконалення програмного рішення.

У результаті функціональні вимоги до системи аналізу логів вебсервера можна узагальнити як сукупність взаємопов’язаних можливостей, спрямованих на автоматизацію обробки журналів подій і виявлення потенційних кібератак. Система повинна забезпечувати завантаження лог-файлу, парсинг записів, нормалізацію даних, фільтрацію технічного шуму, виявлення підозрілих URL, аналіз кодів відповідей, оцінювання частоти запитів, розрахунок рівня ризику, збереження результатів у базі даних, перегляд подій в адміністративній панелі,

фільтрацію та сортування записів, візуалізацію статистики, журналювання спрацювань і тестування на змодельованих сценаріях. Саме ці вимоги створюють основу для наступного підрозділу, у якому буде розглянуто розробку алгоритму обробки логів та оцінювання рівня ризику користувацької активності.

### 2.2. Розробка алгоритму обробки логів та оцінювання рівня ризику

Розробка алгоритму обробки логів вебсервера є одним із ключових етапів створення системи виявлення потенційних кібератак, оскільки саме алгоритм визначає, яким чином сирі записи журналу подій перетворюються на структуровану інформацію, придатну для аналізу, оцінювання ризику та подальшого відображення в адміністративній панелі. На відміну від ручного перегляду логів, який є малоефективним у разі великої кількості записів, алгоритмічна обробка дозволяє автоматизувати основні етапи аналізу: зчитування даних, виділення ключових параметрів, нормалізацію, виявлення підозрілих ознак, групування подій і формування підсумкового рівня ризику. Такий підхід відповідає загальній логіці управління журналами безпеки, де логи розглядаються не лише як технічний архів, а як джерело даних для моніторингу, розслідування інцидентів та підтримки прийняття рішень [37]. NIST SP 800-92 визначає управління журналами як процес, що охоплює генерацію, передавання, зберігання, аналіз і захист логів, що підтверджує необхідність їх системної обробки в межах кібербезпеки.

На початковому етапі алгоритм повинен забезпечити отримання вхідних даних, тобто лог-файлу вебсервера. У межах розроблюваної системи доцільно передбачити можливість завантаження файлу адміністратором через інтерфейс або вибір файлу з визначеної директорії. Джерелом даних найчастіше є access log, оскільки він містить інформацію про звернення клієнтів до вебресурсу. У таких записах зазвичай фіксуються IP-адреса клієнта, дата й час запиту, HTTP-метод, URL, код відповіді сервера, розмір відповіді, referer та User-Agent. Водночас формат логів може відрізнятися залежно від типу вебсервера та його конфігурації. Apache HTTP Server дозволяє гнучко задавати формат логування через LogFormat,

а Nginx підтримує налаштування access_log і log_format, що дає змогу визначати, які саме параметри потрібно записувати до журналу [38]. Документація Apache прямо описує використання форматів журналювання, а Nginx фіксує запити через access logs, зокрема в combined format за типовою конфігурацією.

Після отримання лог-файлу система повинна виконати його попередню перевірку. На цьому етапі доцільно перевірити формат файлу, його розмір, наявність записів і можливість прочитати вміст як текстові дані. Така перевірка потрібна для того, щоб запобігти помилкам під час обробки та не допустити завантаження сторонніх або надмірно великих файлів. При цьому алгоритм не повинен повністю припиняти роботу через окремі некоректні рядки, оскільки реальні журнали подій можуть містити пошкоджені або нестандартні записи. Доцільніше передбачити часткову обробку: коректні рядки аналізуються, а некоректні зберігаються або рахуються окремо як помилки парсингу. Такий підхід робить систему більш стійкою до реальних умов використання.

Наступним етапом є парсинг логів, тобто перетворення кожного текстового рядка на структурований запис. Для системи аналізу важливо виділити з журналу основні параметри: IP-адресу, часову мітку, метод запиту, URL, код відповіді, розмір відповіді та User-Agent. Саме ці поля надалі використовуються для оцінювання підозрілої активності. IP-адреса дозволяє групувати запити за джерелом, часова мітка — визначати інтенсивність дій, URL — виявляти спроби доступу до службових або небезпечних ресурсів, код відповіді — аналізувати помилки доступу, а User-Agent — розпізнавати нетипові клієнтські програми або автоматизовані інструменти. У межах дипломної роботи парсинг можна реалізувати за допомогою регулярного виразу, орієнтованого на типовий формат

access log.

```
import re
log_pattern = re.compile(
   r'(?P<ip>\S+) \S+ \S+ \[(?P<time>[^\]]+)\] '
   r'"(?P<method>\S+) (?P<url>\S+) (?P<protocol>[^"]+)" '
```

```
    r'(?P<status>\d{3}) (?P<size>\S+) '
    r'"(?P<referer>[^"]*)" "(?P<user_agent>[^"]*)"'
)
def parse_log_line(line):
    match = log_pattern.match(line)
    return match.groupdict() if match else None
```

Наведений фрагмент коду демонструє базову логіку перетворення рядка логу на набір структурованих полів. У повній реалізації до цього механізму доцільно додати обробку помилок, підтримку різних форматів журналів і перевірку коректності отриманих значень. Важливо, щоб система не лише формально розбивала рядок, а й готувала дані до подальшого аналізу, оскільки саме від якості парсингу залежить точність визначення потенційно небезпечних подій.

Після парсингу алгоритм має виконати нормалізацію даних. Її зміст полягає у приведенні значень до єдиного формату. Наприклад, HTTP-методи доцільно переводити до верхнього регістру, URL — до нижнього регістру та декодованого вигляду, коди відповідей — до числового типу, а часові мітки — до стандартного формату дати й часу. Нормалізація потрібна тому, що однакові за змістом запити можуть мати різний вигляд через кодування символів, різний регістр або особливості запису. Без цього система може не розпізнати підозрілу активність або, навпаки, створити зайві хибні спрацювання. OWASP Logging Cheat Sheet підкреслює, що записи журналів повинні містити достатньо інформації для подальшого моніторингу й аналізу, але водночас не повинні створювати ризику витоку зайвих чутливих даних [39].

Після нормалізації доцільно виконати попередню класифікацію записів. На цьому етапі система визначає, до якого типу належить запит: звичайний перегляд сторінки, звернення до статичного ресурсу, запит до адміністративної панелі, спроба авторизації, API-запит або потенційно підозріле звернення. Така класифікація допомагає не перевантажувати адміністратора другорядними подіями та водночас не втрачати важливу інформацію. Наприклад, запити до

зображень або CSS-файлів у більшості випадків не є пріоритетними для аналізу, але різке збільшення їх кількості може мати значення під час навантажувальної атаки. Тому система повинна не видаляти такі записи, а лише маркувати їх відповідною категорією.

Ключовою частиною алгоритму є виявлення підозрілих ознак. Для цього доцільно використати поєднання сигнатурного та евристичного підходів. Сигнатурний підхід передбачає пошук відомих шаблонів, які можуть бути пов’язані з SQL-ін’єкціями, XSS-атаками, path traversal, зверненнями до конфігураційних файлів або службових директорій. Евристичний підхід полягає в оцінюванні загальної поведінки клієнта: частоти запитів, кількості помилок, нетипових HTTP-методів, повторюваних звернень до адміністративних сторінок або надмірної активності з однієї IP-адреси. Поєднання цих підходів є доцільним, оскільки одиничний шаблон не завжди доводить атаку, але сукупність ознак може свідчити про підвищений рівень ризику.

Окрему увагу в алгоритмі потрібно приділити аналізу HTTP-кодів відповідей. Коди 401 і 403 можуть свідчити про спроби доступу до захищених ресурсів, 404 — про звернення до неіснуючих сторінок або сканування директорій, 405 — про використання непідтримуваного методу, 429 — про перевищення допустимої частоти запитів, а коди 5xx можуть вказувати на перевантаження або помилки на стороні сервера. При цьому важливо аналізувати не лише сам факт появи певного коду, а й частоту його повторення. Один запис із кодом 404 може бути звичайною помилкою користувача, однак десятки або сотні таких записів за короткий час із однієї IP-адреси можуть свідчити про автоматизоване сканування вебресурсу.

Важливим елементом алгоритму є аналіз частоти запитів. Багато кібератак проявляються саме через повторюваність дій: brute force — через велику кількість спроб авторизації, сканування — через часті звернення до різних URL, HTTP flood — через надмірну кількість однотипних запитів. Тому система повинна групувати записи за IP-адресами та часовими проміжками, наприклад за одну хвилину або п’ять хвилин. Якщо кількість запитів перевищує встановлений поріг, система

повинна підвищити рівень ризику для відповідної активності. Порогові значення мають бути налаштовуваними, оскільки нормальна активність для невеликого сайту та великого вебсервісу може суттєво відрізнятися.

Основою оцінювання ризику в межах цієї роботи є бальна модель. Її перевага полягає в простоті, прозорості та можливості пояснити результат. Кожна підозріла ознака додає певну кількість балів до загального показника. Наприклад, підозрілий URL може додавати два бали, помилка доступу — один або два бали, звернення до адміністративної сторінки — два бали, велика кількість запитів за короткий час — три бали. Після підсумовування система визначає рівень ризику: низький, середній або високий. Такий підхід є зручним для дипломної роботи, оскільки дозволяє наочно показати логіку прийняття рішення та не потребує складних моделей машинного навчання.

```
def calculate_risk(event, request_count):
    score = 0
    reasons = []
    suspicious_patterns = ["union", "select", "<script", "../", ".env", "admin", "wp-login"]
    if any(pattern in event["url"].lower() for pattern in suspicious_patterns):
        score += 2
        reasons.append("виявлено підозрілий шаблон у URL")
    if event["status"] in [401, 403, 404]:
        score += 1
        reasons.append("зафіксовано помилку доступу або неіснуючий ресурс")
    if event["method"] not in ["GET", "POST", "HEAD"]:
        score += 2
        reasons.append("використано нетиповий HTTP-метод")
    if request_count > 100:
        score += 3
        reasons.append("надмірна кількість запитів з однієї IP-адреси")
    if score >= 6:
        level = "high"
```

```
elif score >= 3:
    level = "medium"
else:
    level = "low"
return score, level, reasons
```

Цей фрагмент коду демонструє спрощену модель оцінювання ризику, яку можна використати як основу для програмної реалізації. Важливо, що функція повертає не лише кількість балів і рівень ризику, а й пояснення, тобто причини, через які подія була позначена як підозріла. Така пояснюваність є важливою для адміністратора, оскільки дає змогу швидко зрозуміти, чи потребує подія додаткової перевірки.

У межах алгоритму доцільно передбачити три рівні ризику. Низький рівень встановлюється для звичайних подій або записів із мінімальними відхиленнями, які не потребують негайної реакції. Середній рівень присвоюється подіям, які містять одну або кілька підозрілих ознак, але не мають критичної інтенсивності. Високий рівень ризику встановлюється у випадках, коли поєднуються кілька небезпечних факторів: підозрілий URL, часті помилки доступу, надмірна кількість запитів, звернення до адміністративної сторінки або використання нетипового HTTP-методу. Такий поділ дозволяє адміністратору швидко зосередитися на найбільш важливих подіях і не витрачати час на перегляд усіх записів однаково.

Логіку оцінювання ризику можна подати у вигляді таблиці критеріїв, яка використовується під час роботи алгоритму.

Таблиця 2.1

| **Ознака** | **Приклад прояву** | **Оцінка впливу** |
|---|---|---|
| Підозрілий URL | union, select, script, ../, .env | +2 бали |
| Помилка доступу | 401, 403, 404 | +1 бал |
| Нетиповий HTTP-метод | PUT, DELETE, OPTIONS, PATCH | +2 бали |
| Адміністративний ресурс | admin, wp-login, manager | +2 бали |
| Висока частота запитів | понад 100 запитів за короткий проміжок | +3 бали |
| Порожній або нетиповий User-Agent | відсутній або дуже короткий рядок | +1 бал |

| Ознака | Приклад прояву | Оцінка впливу |
|---|---|---|
| Помилки сервера | 500, 502, 503, 504 | +2 бали |

Після оцінювання ризику система повинна зберігати результати обробки у базі даних. До кожного запису доцільно додавати не лише вихідні параметри логу, а й додаткові поля: тип ресурсу, кількість балів ризику, рівень ризику, причини спрацювання та час обробки. Це дозволить надалі швидко фільтрувати події в адміністративній панелі, переглядати лише високоризикові записи, будувати статистику та аналізувати активність окремих IP-адрес. Збереження пояснень також важливе для тестування системи, оскільки дає змогу перевірити, чи правильно алгоритм класифікує змодельовані сценарії атак.

Важливою частиною алгоритму є зменшення кількості хибних спрацювань. Не кожна підозріла ознака означає реальну атаку. Наприклад, помилка 404 може виникати через застаріле посилання, а велика кількість запитів може належати легітимному пошуковому роботу або сервісу моніторингу. Тому алгоритм повинен враховувати не одну ізольовану ознаку, а їх сукупність. Для зменшення хибних спрацювань можна передбачити список довірених IP-адрес, налаштування порогів частоти, окрему обробку відомих пошукових ботів і можливість ручного перегляду причин спрацювання. Саме такий підхід дозволяє зробити систему не лише технічно працездатною, а й практично зручною.

Окремо слід враховувати безпечне відображення результатів аналізу. Оскільки в логах можуть міститися фрагменти шкідливих запитів, наприклад XSS-код у URL або User-Agent, система не повинна виводити ці дані в адміністративній панелі без екранування. Інакше може виникнути ситуація, коли шкідливий фрагмент, записаний у лог, виконається вже під час перегляду журналу адміністратором. OWASP Poor Logging Practice звертає увагу, що неправильне логування може створювати додаткові ризики, зокрема витік інформації, log injection або ускладнення реагування на інциденти [41].

Таким чином, алгоритм обробки логів та оцінювання рівня ризику має складатися з кількох взаємопов’язаних етапів: отримання лог-файлу, перевірки

його коректності, парсингу записів, нормалізації даних, попередньої класифікації подій, виявлення підозрілих ознак, аналізу HTTP-кодів, оцінювання частоти запитів, розрахунку рівня ризику, збереження результатів і відображення їх в адміністративній панелі. Основою алгоритму є rule-based і risk-score підхід, який дозволяє прозоро визначати рівень потенційної небезпеки кожної події. Перевагами такого підходу є зрозумілість, простота реалізації, можливість пояснення результатів і придатність для подальшого розширення. У майбутньому алгоритм може бути доповнений машинним навчанням, інтеграцією з SIEM, WAF або зовнішніми індикаторами компрометації, однак для меж цієї роботи обраний підхід є достатнім для реалізації базової системи виявлення потенційних кібератак на основі аналізу логів вебсервера.

### 2.3. Проєктування архітектури системи та структури бази даних

Розробка алгоритму оцінювання рівня ризику користувацької активності є одним із центральних елементів вебсистеми моніторингу та виявлення підозрілої активності, оскільки саме цей алгоритм перетворює звичайний журнал подій на аналітичний інструмент, здатний автоматично визначати, які дії користувача є типовими, а які потребують уваги адміністратора. Якщо модуль журналювання лише фіксує факт події, наприклад вхід до системи, невдалу спробу авторизації, звернення до забороненого ресурсу або зміну ролі, то модуль оцінювання ризику встановлює, наскільки така подія може бути небезпечною в конкретному контексті. У межах цієї дипломної роботи доцільно застосувати rule-based підхід, тобто алгоритм на основі правил, оскільки він є прозорим, зручним для реалізації, добре піддається тестуванню та дозволяє пояснити, чому конкретна подія отримала певний рівень ризику. Такий підхід узгоджується з логікою OWASP Risk Rating Methodology, де ризик розглядається через поєднання ймовірності та впливу, а результат може бути поданий у вигляді рівня серйозності.

У загальному вигляді алгоритм оцінювання ризику користувацької активності повинен виконувати кілька послідовних дій: отримувати вхідні дані про подію, визначати її тип, аналізувати контекст, перевіряти набір правил, нараховувати бали

ризику, встановлювати рівень ризику, зберігати результат у журналі та передавати його до адміністративної панелі. Вхідними даними для алгоритму є параметри, які вже фіксуються системою журналювання: user_id, event_type, ip_address, user_agent, endpoint, http_method, status_code, created_at, кількість подібних подій за певний період, інформація про роль користувача та додаткові ознаки, наприклад факт входу з нової IP-адреси або спроба доступу до адміністративного маршруту [48]. Саме якість цих даних визначає ефективність алгоритму, оскільки без часової мітки неможливо оцінити частоту дій, без IP-адреси неможливо визначити нове джерело входу, без типу події неможливо класифікувати активність, а без ролі користувача неможливо зрозуміти, чи мав він право виконувати відповідну дію. NIST SP 800-92 підкреслює значення управління журналами для комп'ютерної безпеки, зокрема їх збирання, зберігання, аналізу та використання для виявлення проблем безпеки.

Алгоритм оцінювання рівня ризику доцільно будувати не як жорстке блокування всіх нетипових дій, а як механізм попередньої класифікації подій за рівнем небезпечності. Це важливо, оскільки не кожна нетипова дія є реальною атакою або інцидентом. Наприклад, одна невдала спроба входу може бути звичайною помилкою користувача, вхід з нової IP-адреси може бути наслідком роботи з дому, а доступ у неробочий час може бути пов'язаний із терміновим завданням. Водночас поєднання кількох ознак, наприклад багаторазові невдалі спроби входу, подальша успішна авторизація, нова IP-адреса, звернення до адміністративного endpoint та спроба завантаження великої кількості файлів, уже формує значно вищий рівень ризику. Саме тому алгоритм має працювати з накопичувальною системою балів, де кожна окрема ознака додає певну кількість балів, а підсумкове значення визначає загальний рівень ризику.

У межах розроблюваної вебсистеми пропонується використовувати три рівні ризику: низький, середній і високий. Низький рівень ризику встановлюється для звичайних або слабко підозрілих подій, які не потребують термінового реагування, але мають зберігатися в журналі. Середній рівень ризику встановлюється для подій, які можуть свідчити про потенційне порушення

політики доступу або нетипову поведінку користувача. Високий рівень ризику встановлюється для подій або комбінацій подій, які можуть свідчити про компрометацію облікового запису, спробу несанкціонованого доступу, підбір пароля, зловживання правами або іншу небезпечну активність. Така трирівнева модель є зручною для адміністративної панелі, оскільки вона дозволяє швидко відокремити критичні події від звичайних записів журналу, не перевантажуючи адміністратора надмірною кількістю складних категорій.

Для практичної реалізації алгоритму доцільно застосувати числову шкалу від 0 до 100 балів. Події з показником від 0 до 30 балів можна віднести до низького рівня ризику, від 31 до 70 балів — до середнього, а понад 70 балів — до високого. Такий підхід є зрозумілим і легко реалізується в коді, оскільки система може спочатку підрахувати risk_score, а потім автоматично визначити risk_level. Крім того, числова шкала дозволяє гнучко змінювати правила в майбутньому, наприклад підвищити вагу невдалих входів, додати окрему оцінку для масового завантаження файлів або знизити ризик певних подій для довірених IP-адрес. OWASP Logging Cheat Sheet звертає увагу на важливість фіксації подій автентифікації, авторизації, помилок доступу, змін прав і важливих операцій, що прямо відповідає набору подій, які мають враховуватися в алгоритмі.

Система правил алгоритму повинна включати базові, контекстні та кореляційні ознаки. Базові ознаки пов’язані з типом події: невдала спроба входу, заборонений доступ, зміна пароля, зміна ролі, спроба виконати адміністративну дію, серверна помилка, звернення до API або завантаження файлу. Контекстні ознаки враховують обставини виконання дії: нова IP-адреса, незвичний user-agent, неробочий час, критичний endpoint, небезпечний HTTP-метод, статус-код 401 або 403, відсутність необхідної ролі. Кореляційні ознаки визначаються через аналіз кількох подій у межах певного часового інтервалу: кількість невдалих входів за 10 хвилин, кількість запитів за 1 хвилину, кількість звернень до заборонених ресурсів за добу, кількість завантажень файлів за короткий період. Саме кореляційні ознаки роблять алгоритм більш практичним, оскільки підозріла активність часто проявляється не в одній дії, а в послідовності дрібних подій.

Орієнтовна система нарахування балів може бути подана таким чином:

| Ознака користувацької активності | Умова спрацювання | Кількість балів |
|---|---|---|
| Невдала спроба входу | event_type = login_failed | +10 |
| Серія невдалих входів | 5 і більше невдалих входів за 10 хвилин | +40 |
| Заборонений доступ | HTTP 403 або forbidden_access | +30 |
| Неавторизований запит | HTTP 401 або відсутній токен | +20 |
| Доступ до адміністративного endpoint | Запит до /api/admin або /api/users/manage | +25 |
| Небезпечний HTTP-метод | DELETE, PUT, PATCH | +15 |
| Вхід із нової IP-адреси | IP-адреса не зустрічалася раніше для користувача | +20 |
| Активність у неробочий час | Подія після 22:00 або до 06:00 | +15 |
| Надмірна частота API-запитів | Понад 100 запитів за 1 хвилину | +50 |
| Масове завантаження файлів | Понад 20 завантажень за короткий інтервал | +60 |

Наведена система балів не є універсальною для всіх організацій, але вона є придатною для дипломного програмного продукту, оскільки демонструє принцип формалізації ризику. У реальному корпоративному середовищі ваги правил можуть бути змінені залежно від політики безпеки, типу інформації, рівня критичності ресурсу, робочого графіка користувачів і допустимого рівня хибних спрацювань. Для дипломної роботи важливо показати, що алгоритм є налаштовуваним: адміністратор або розробник може змінити порогові значення, додати нові правила або змінити вагу окремих факторів. У подальшому такі правила можна винести в окрему таблицю risk_rules, але в першій версії системи їх достатньо реалізувати в окремому модулі RiskEngine.

Важливим елементом алгоритму є оцінювання не лише типу події, а й ролі користувача. Однакова дія може мати різний рівень ризику залежно від того, хто її виконує. Наприклад, звернення до сторінки керування користувачами є нормальною дією для адміністратора, але для звичайного користувача така дія має вважатися підозрілою. Аналогічно, метод DELETE може бути допустимим для адміністратора в межах окремого модуля, але для користувача без відповідних прав він повинен збільшувати ризик. OWASP ASVS розглядає перевірку контролю доступу як одну з важливих частин безпеки застосунку, тому алгоритм оцінювання ризику має бути пов’язаний із role-based access control, а не працювати ізольовано від авторизації.

Алгоритм має також враховувати часовий контекст. Для цього доцільно використовувати часові вікна, наприклад 1 хвилина, 10 хвилин, 1 година та 24 години. Коротке вікно в 1 хвилину зручно використовувати для виявлення надмірної кількості API-запитів, 10-хвилинне вікно — для аналізу серії невдалих входів, годинне вікно — для виявлення повторюваних заборонених дій, а добове вікно — для аналізу загальної активності користувача. На рівні бази даних це означає, що таблиця security_events повинна мати поле created_at, а також індекс за ним, щоб система могла швидко виконувати запити на підрахунок подій за певний період. PostgreSQL підтримує індекси, які застосовуються для підвищення продуктивності запитів до таблиць, що особливо важливо для журналів подій, які можуть швидко накопичуватися.

З погляду практичної реалізації алгоритм можна подати як окрему функцію або клас, який отримує об’єкт події й повертає три значення: числовий бал ризику, текстовий рівень ризику та причини спрацювання правил. Причини є важливими, оскільки адміністратор повинен бачити не лише те, що подія отримала високий ризик, а й чому саме це сталося. Наприклад, у деталях події можна показати: багаторазові невдалі спроби входу за 10 хвилин, нова IP-адреса, заборонений endpoint. Це робить систему прозорою й підвищує довіру до її результатів. Невеликий фрагмент коду алгоритму може мати такий вигляд:

```
    def calculate_risk(event, context):
score = 0
reasons = []

if event.event_type == "login_failed":
    score += 10
    reasons.append("Невдала спроба входу")

if context.failed_logins_10min >= 5:
    score += 40
    reasons.append("5 або більше невдалих входів за 10 хвилин")

if event.status_code == 403:
    score += 30
    reasons.append("Спроба доступу до забороненого ресурсу")

if event.endpoint.startswith("/api/admin"):
    score += 25
    reasons.append("Звернення до адміністративного endpoint")

if context.is_new_ip:
    score += 20
    reasons.append("Вхід із нової IP-адреси")

if score >= 70:
    level = "high"
elif score >= 31:
    level = "medium"
else:
    level = "low"
```

```
return score, level, reasons
```

Цей код не є повною реалізацією всієї системи, але він демонструє основну ідею алгоритму. Об’єкт event містить дані поточної події, а об’єкт context містить додаткову інформацію, отриману з бази даних: кількість невдалих входів за останні 10 хвилин, факт нової IP-адреси, кількість запитів за хвилину, роль користувача та інші показники. Такий поділ є важливим, оскільки ризик не можна повноцінно оцінити лише за однією подією. Алгоритм повинен бачити контекст, інакше він буде реагувати лише на очевидні випадки, пропускаючи складніші сценарії.

У структурі бази даних результати роботи алгоритму доцільно зберігати безпосередньо в таблиці security_events. Для цього використовуються поля risk_score, risk_level і, за потреби, risk_reasons. Поле risk_score зберігає числовий бал, risk_level — текстове значення low, medium або high, а risk_reasons може зберігати перелік причин у форматі JSON або тексту. Якщо використовується PostgreSQL, для такого поля можна застосувати JSONB, що дозволить зберігати масив причин і зручно відображати його в інтерфейсі. Наприклад, подія може мати risk_score = 85, risk_level = high, а risk_reasons = ["5 або більше невдалих входів за 10 хвилин", "Нова IP-адреса", "Доступ до адміністративного endpoint"]. Така деталізація допомагає адміністратору швидко зрозуміти логіку спрацювання.

Алгоритм оцінювання ризику також повинен бути пов’язаний із життєвим циклом події. Після створення подія отримує статус new, після перегляду адміністратором — in_review, після підтвердження — confirmed, після завершення обробки — resolved, а в разі помилкового спрацювання — false_positive. Це важливо, оскільки навіть правильно налаштований алгоритм може створювати хибні спрацювання. Наприклад, користувач може випадково кілька разів помилитися з паролем або зайти з нової IP-адреси під час відрядження. Тому система не повинна автоматично трактувати кожну високоризикову подію як доведений інцидент. Вона повинна формувати пріоритет для перевірки, а остаточне рішення може приймати адміністратор.

Для підвищення якості алгоритму доцільно використовувати кореляцію подій. Кореляція означає, що система аналізує не лише поточну дію, а й попередні події, пов’язані з тим самим користувачем, IP-адресою або endpoint. Наприклад, якщо система бачить три невдалі входи, потім успішний вхід, а після цього зміну пароля та масове завантаження файлів, загальний рівень ризику повинен бути вищим, ніж у кожної окремої події. MITRE ATT&CK використовується як загальнодоступна база знань про тактики й техніки кіберзагроз, заснована на реальних спостереженнях, і її логіка добре показує, що небезпечна активність часто складається з послідовності дій, а не з одного ізольованого запиту.

Водночас алгоритм повинен бути достатньо простим, щоб його можна було реалізувати в межах дипломної роботи. Тому на цьому етапі недоцільно використовувати складні моделі машинного навчання, нейронні мережі або поведінкові профілі, які потребують великого набору реальних даних. Rule-based алгоритм має перевагу в тому, що його можна протестувати на змодельованих сценаріях. Наприклад, можна створити тестового користувача, виконати кілька невдалих входів, потім спробувати зайти до адміністративної сторінки, після чого перевірити, чи система правильно присвоїла події середній або високий рівень ризику. Такий підхід є достатнім для демонстрації принципу роботи системи й відповідає реалістичним вимогам до дипломного програмного продукту.

Алгоритм повинен бути інтегрований у загальну архітектуру вебсистеми. Після кожної значущої події backend формує об’єкт події, передає його до RiskEngine, отримує risk_score, risk_level і risk_reasons, після чого зберігає результат у таблиці security_events. Потім ці дані стають доступними через API для frontend-частини. На сторінці DashboardPage адміністратор бачить кількість високоризикових подій, на сторінці EventsPage може фільтрувати журнал за рівнем ризику, а на сторінці деталей події бачить причини нарахування балів. FastAPI підтримує створення API-маршрутів і захист endpoint за допомогою механізмів OAuth2 та JWT, що дає змогу поєднати алгоритм ризику з автентифікацією й авторизацією користувачів.

З точки зору моніторингу важливо, щоб алгоритм не лише класифікував окремі події, а й формував дані для загальної аналітики. На основі risk_score можна підраховувати середній ризик користувача за добу, кількість високоризикових подій за останній тиждень, IP-адреси з найбільшою кількістю підозрілих дій, endpoint з найбільшою кількістю заборонених звернень, а також динаміку зміни ризику у часі. Такі показники відображаються у dashboard і допомагають адміністратору швидко оцінити стан системи. NIST Cybersecurity Framework 2.0 визначає функцію Detect як одну з ключових функцій кібербезпеки поряд із Govern, Identify, Protect, Respond і Recover, що підтверджує значення своєчасного виявлення аномалій і подій безпеки.

Окрему увагу потрібно приділити зменшенню кількості хибних спрацювань. Для цього в алгоритмі можна передбачити довірені IP-адреси, винятки для службових endpoint, різні ваги для ролей користувачів, а також зміну порогових значень. Наприклад, для адміністратора доступ до /api/admin не повинен автоматично збільшувати ризик, якщо він має відповідний дозвіл, але для звичайного користувача така дія має бути ризиковою. Також варто уникати ситуації, коли кожен запит зі статусом 404 відразу вважається загрозою, оскільки частина таких подій може бути звичайною помилкою навігації. Тому алгоритм має оцінювати не тільки сам факт помилки, а й її частоту, повторюваність і зв’язок з іншими діями.

Для перевірки роботи алгоритму необхідно створити набір тестових сценаріїв. Перший сценарій може передбачати звичайну активність користувача: успішний вхід, перегляд дозволеної сторінки, вихід із системи; такий сценарій має створювати події з низьким ризиком. Другий сценарій може передбачати кілька невдалих спроб входу; система має підвищити ризик до середнього. Третій сценарій може включати п’ять і більше невдалих входів за 10 хвилин; система повинна встановити високий рівень ризику [49]. Четвертий сценарій може передбачати спробу звичайного користувача звернутися до адміністративного endpoint; система має зафіксувати forbidden_access. П’ятий сценарій може поєднувати нову IP-адресу, неробочий час і доступ до критичного ресурсу; така

комбінація має формувати високий ризик. CIS Controls v8 окремо виділяє audit log management як важливий контроль, що підтверджує значення збирання та аналізу журналів для практичного виявлення подій безпеки.

Результати тестування алгоритму доцільно подати у вигляді таблиці, де зазначається сценарій, очікуваний результат і фактичний рівень ризику. Це дозволить продемонструвати, що алгоритм працює не декларативно, а реально класифікує події. Наприклад, для сценарію звичайного входу очікується low, для серії невдалих входів — medium або high, для спроби доступу до адміністративного ресурсу без прав — high, для масового API-навантаження — high. Така таблиця буде корисною в третьому розділі, де описуватиметься тестування програмної реалізації.

У межах розроблюваної вебсистеми алгоритм оцінювання рівня ризику користувацької активності виконує роль аналітичного ядра. Його завдання полягає не в тому, щоб повністю замінити адміністратора або професійну SIEM-систему, а в тому, щоб автоматично впорядкувати журнал подій за рівнем важливості. Це особливо важливо для невеликих корпоративних середовищ або навчального прототипу, де немає складної інфраструктури кібербезпеки, але є потреба швидко бачити потенційно небезпечні дії. Закон України Про основні засади забезпечення кібербезпеки України визначає правові та організаційні основи забезпечення кібербезпеки, що створює загальний нормативний контекст для розробки програмних засобів моніторингу, виявлення та реагування на кіберінциденти.

Отже, запропонований алгоритм оцінювання рівня ризику користувацької активності ґрунтується на поєднанні журналювання, контекстного аналізу, системи правил, числової шкали ризику, класифікації подій і подальшої візуалізації результатів у адміністративній панелі. Основними вхідними параметрами алгоритму є тип події, статус-код, endpoint, HTTP-метод, IP-адреса, роль користувача, час події та кількість подібних дій у межах визначеного періоду. Основним результатом є risk_score, risk_level і перелік причин, які пояснюють спрацювання правил. Така модель є достатньо простою для реалізації в межах дипломної роботи, але водночас достатньо змістовною, щоб показати практичний

зв’язок між теорією інформаційної безпеки, архітектурою вебзастосунку, журналюванням і програмним аналізом підозрілої активності.

**Висновки до розділу 2**

У другому розділі було розглянуто питання проєктування системи аналізу логів вебсервера для виявлення потенційних кібератак. На основі теоретичних положень, розкритих у першому розділі, було визначено прикладну логіку майбутньої системи, її основні функціональні можливості, послідовність обробки журналів подій та підхід до оцінювання рівня ризику. Встановлено, що ефективна система аналізу логів повинна не лише зчитувати записи журналу, а й перетворювати їх на структуровані дані, придатні для подальшого пошуку підозрілих ознак, групування подій, фільтрації, статистичного аналізу та візуального представлення результатів.

У межах розділу було сформовано функціональні вимоги до системи аналізу логів. До основних вимог віднесено можливість завантаження лог-файлу, перевірку його коректності, парсинг записів, нормалізацію даних, фільтрацію технічного шуму, виявлення підозрілих URL, аналіз HTTP-кодів відповідей, оцінювання частоти запитів, визначення рівня ризику, збереження результатів у базі даних, перегляд подій в адміністративній панелі та побудову статистичних показників. Такий підхід відповідає загальній логіці управління журналами безпеки, оскільки NIST розглядає log management як процес, що охоплює збирання, зберігання, аналіз і захист журналів подій, а не лише їх пасивне накопичення.

Особливу увагу було приділено алгоритму обробки логів і формуванню інтегрального рівня ризику. Запропонований алгоритм передбачає послідовне виконання кількох етапів: отримання лог-файлу, перевірку його структури, парсинг рядків, нормалізацію полів, попередню класифікацію запитів, виявлення підозрілих шаблонів, аналіз кодів відповідей, оцінювання частоти звернень з окремих IP-адрес і формування підсумкової оцінки ризику. Такий порядок дає

змогу перейти від неструктурованого текстового файлу до зрозумілого набору подій, кожна з яких має визначені параметри, рівень потенційної небезпеки та пояснення причин спрацювання.

У розділі обґрунтовано доцільність використання rule-based і risk-score підходу. Його перевага полягає в тому, що система працює на основі зрозумілих правил, а кожна підозріла ознака додає певну кількість балів до загальної оцінки. Наприклад, підозрілий URL, помилки доступу, нетиповий HTTP-метод, часті запити з однієї IP-адреси або звернення до адміністративних сторінок можуть підвищувати рівень ризику події. На відміну від складних моделей машинного навчання, така логіка є прозорою, легко пояснюється в межах дипломної роботи та дозволяє адміністратору бачити не лише кінцеву оцінку, а й конкретні причини, через які подія була позначена як потенційно небезпечна.

Також було визначено, що система повинна враховувати проблему хибних спрацювань. Одинична помилка 404, випадкова спроба входу або велика кількість запитів від легітимного пошукового робота не завжди свідчать про кібератаку. Саме тому оцінювання ризику має ґрунтуватися не на одній ізольованій ознаці, а на сукупності параметрів: URL, код відповіді, HTTP-метод, частота звернень, User-Agent, тип ресурсу та часовий контекст. OWASP підкреслює, що якісне логування має бути пов’язане з моніторингом, розслідуванням інцидентів і реагуванням на події безпеки, тому результати аналізу мають бути придатними для подальшої практичної перевірки адміністратором.

У процесі проєктування було встановлено, що важливим компонентом системи є адміністративна панель. Вона повинна забезпечувати перегляд оброблених логів, сортування подій за рівнем ризику, фільтрацію за IP-адресою, URL, кодом відповіді, датою або типом події, а також відображення причин спрацювання алгоритму. Практичне значення такого інтерфейсу полягає в тому, що адміністратор отримує не просто великий масив технічних записів, а впорядковану аналітичну інформацію, яка дозволяє швидко визначити найбільш підозрілі події та прийняти рішення щодо подальшої перевірки.

Окремо було наголошено на необхідності безпечної обробки та відображення логів. Оскільки самі записи журналів можуть містити підозрілі URL, фрагменти XSS-коду, нестандартні User-Agent або інші потенційно небезпечні значення, система не повинна виводити такі дані без належного екранування. Неналежна практика логування може ускладнювати централізований моніторинг, кореляцію подій, виявлення інцидентів і подальший аналіз, на що звертає увагу OWASP у матеріалах щодо проблем поганого логування.

Отже, у другому розділі було сформовано прикладну основу для подальшої програмної реалізації системи аналізу логів вебсервера. Визначені функціональні вимоги й алгоритм обробки даних дозволяють побудувати систему, здатну автоматизовано аналізувати журнали подій, виявляти потенційно підозрілу активність, оцінювати рівень ризику та відображати результати у зручному для адміністратора вигляді. Запропонований підхід є достатнім для реалізації в межах дипломної роботи та водночас може бути розширений у майбутньому шляхом інтеграції з SIEM, WAF, зовнішніми індикаторами компрометації, системами сповіщень або модулями машинного навчання.

# РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ АНАЛІЗУ ЛОГІВ ВЕБСЕРВЕРА

## 3.1. Обґрунтування вибору технологій та реалізація основних функціональних компонентів

Програмна реалізація системи аналізу логів вебсервера для виявлення потенційних кібератак потребує обґрунтованого вибору технологій, оскільки саме технологічна основа визначає стабільність роботи системи, зручність її подальшого розширення, швидкість обробки журналів подій, можливість збереження результатів аналізу та доступність інтерфейсу для адміністратора. У межах цієї роботи доцільно орієнтуватися не на надмірно складну промислову інфраструктуру, а на зрозумілий, відтворюваний і достатньо надійний стек технологій, який дозволяє реалізувати основні функції системи: завантаження лог-файлу, парсинг записів, нормалізацію даних, оцінювання рівня ризику, збереження результатів у базі даних, перегляд подій в адміністративній панелі та формування статистичних показників. Такий підхід відповідає загальній логіці інженерії програмного забезпечення, згідно з якою вибір технологій має визначатися не лише популярністю інструментів, а їх відповідністю функціональним вимогам, структурі даних, очікуваному навантаженню, складності підтримки та можливості подальшої модернізації.

З наукової точки зору розробка такої системи спирається на ідею спостережності програмних систем, тобто здатності отримувати з технічних даних інформацію про стан, поведінку та потенційні проблеми системи. У сучасних дослідженнях логування розглядається не як другорядний технічний процес, а як складова ширшої концепції observability, яка поєднує логи, метрики й трасування для розуміння поведінки програмного середовища. Дослідники A. Taher та інші в межах систематичного картографування log-based software monitoring підкреслюють, що аналіз логів є важливим напрямом у програмній інженерії, однак він залишається складним через великий обсяг даних, неоднорідність форматів і потребу у контекстній інтерпретації подій [43]. Це безпосередньо стосується теми роботи, оскільки система аналізу логів вебсервера повинна не

просто зберігати записи журналу, а перетворювати їх на зрозумілу аналітичну інформацію для виявлення потенційних кібератак.

Для реалізації основних функціональних компонентів доцільно використати мову програмування Python. Її вибір обґрунтовується кількома чинниками. Python має простий і зрозумілий синтаксис, розвинену стандартну бібліотеку, широку підтримку роботи з текстовими файлами, регулярними виразами, датами, базами даних і вебфреймворками. Для системи аналізу логів це особливо важливо, оскільки основні операції пов’язані з обробкою текстових рядків, виділенням структурованих полів, перевіркою шаблонів, підрахунком статистики та формуванням результатів. У стандартній документації Python окремо представлено модулі re для регулярних виразів, sqlite3 для роботи з SQLite, logging для журналювання подій і csv для роботи з табличними файлами, що робить цю мову придатною для створення прикладної системи аналізу даних без необхідності одразу залучати надмірно складні зовнішні інструменти [44]. Офіційна документація Python підтверджує наявність стандартних засобів для роботи з регулярними виразами, базами SQLite та журналюванням, що є важливим для обраної теми.

Вибір Python також має науково-практичне обґрунтування у сфері кібербезпеки. У багатьох дослідженнях і прикладних розробках Python використовується як мова для створення інструментів аналізу даних, автоматизації безпекових перевірок, побудови прототипів систем виявлення загроз і обробки великих масивів технічної інформації. Зокрема, у вітчизняній науковій публікації, присвяченій розробці системи виявлення кіберзагроз на основі аналізу даних з вебресурсів мовою Python, наголошується на придатності цієї мови для реалізації аналітичних процедур у сфері кібербезпеки [45]. Це дозволяє обґрунтувати Python як доцільний інструмент не лише з погляду простоти програмування, а й з позиції відповідності предметній області дослідження.

Для створення вебінтерфейсу системи може бути використаний легкий вебфреймворк Flask або подібний за призначенням інструмент. Вибір Flask є обґрунтованим у разі створення відносно невеликої системи з адміністративною

панеллю, кількома маршрутами, формою завантаження файлу, сторінкою перегляду результатів і статистичними блоками. На відміну від більш комплексних фреймворків, Flask не нав’язує жорсткої структури проєкту, що дає змогу гнучко організувати логіку застосунку відповідно до потреб дипломної роботи. Його можна використовувати для обробки HTTP-запитів, маршрутизації, роботи з шаблонами HTML, приймання завантажених файлів і виведення результатів аналізу в адміністративному інтерфейсі. З погляду програмної інженерії такий вибір відповідає принципу достатності: технологія повинна закривати потреби проєкту без створення зайвої архітектурної складності.

Для збереження результатів аналізу доцільно використати SQLite. Ця база даних добре підходить для навчальної або дипломної системи, оскільки не потребує окремого серверного процесу, складної конфігурації чи адміністрування. SQLite є самодостатнім, безсерверним і транзакційним SQL-рушієм, що прямо зазначено в офіційній документації проєкту [46]. Для системи аналізу логів це має практичне значення, оскільки всі результати обробки можна зберігати в одному локальному файлі бази даних, а сама система залишатиметься простою для запуску, перенесення й тестування. У базі даних доцільно створити таблицю подій, у якій зберігатимуться IP-адреса, дата й час запиту, HTTP-метод, URL, код відповіді, User-Agent, рівень ризику, кількість балів і пояснення причин спрацювання. Така структура забезпечує можливість фільтрації, сортування, побудови статистики та повторного перегляду результатів без необхідності повторно аналізувати вихідний лог-файл.

Якщо система розширюватиметься, доцільним може бути використання SQLAlchemy як інструменту об’єктно-реляційного відображення. SQLAlchemy у власній документації визначається як SQL toolkit і Object Relational Mapper для Python, що надає розробникам гнучкі можливості роботи з SQL і програмними об’єктами [47]. У межах базової дипломної реалізації можна обійтися стандартним sqlite3, однак SQLAlchemy має переваги для подальшого розвитку системи, оскільки дозволяє зручніше описувати моделі даних, змінювати тип бази даних, підтримувати складніші запити та зменшувати залежність бізнес-логіки від

конкретної СУБД. Саме тому в роботі можна зазначити, що базова реалізація орієнтована на SQLite, а використання ORM може розглядатися як перспективний напрям удосконалення.

Окремого обґрунтування потребує вибір підходу до обробки логів. У системі доцільно застосувати модульну структуру, за якої кожний функціональний компонент виконує окреме завдання. Перший модуль відповідає за завантаження лог-файлу, другий — за парсинг рядків, третій — за нормалізацію даних, четвертий — за виявлення підозрілих ознак, п’ятий — за розрахунок рівня ризику, шостий — за збереження результатів, а сьомий — за відображення подій в адміністративній панелі. Така структура відповідає загальним принципам програмної інженерії, зокрема розділенню відповідальності, повторному використанню коду та спрощенню тестування. Якщо кожний компонент має чітко визначену функцію, систему легше перевіряти, розширювати та змінювати без порушення роботи інших частин.

З позиції дослідників у сфері моніторингу програмних систем важливо, щоб логування і подальший аналіз не були ізольованим технічним елементом, а вбудовувалися в загальну архітектуру системи. У багато-кейсовому дослідженні structured and unified logging зазначається, що логи є частиною цілісної спостережності, а структуроване логування допомагає створювати більш узгоджену картину стану програмної системи [48]. Для цієї дипломної роботи це означає, що система повинна працювати не з хаотичним набором рядків, а зі структурованими подіями, які мають однаковий набір полів, можуть бути збережені в базі даних і використані для статистичного аналізу. Саме тому компонент парсингу є не другорядним, а одним із базових елементів реалізації.

Реалізацію функціонального компонента завантаження файлу доцільно виконати через вебформу адміністративної панелі. Користувач обирає лог-файл, після чого система перевіряє його розширення, розмір і можливість прочитати вміст. Така перевірка має значення не лише для зручності, а й для безпеки. Оскільки система приймає файл від користувача, потрібно обмежити типи файлів, не дозволяти виконання завантаженого вмісту та зберігати файл у контрольованій

директорії. У разі помилки система повинна повідомляти користувача про причину відхилення файлу. У разі успішного завантаження файл передається до модуля обробки. Такий механізм є достатнім для дипломного проєкту й одночасно демонструє дотримання базових вимог безпечної роботи із завантажуваними даними.

Компонент парсингу реалізує перетворення кожного рядка журналу в об’єкт події. Для цього може використовуватися регулярний вираз, орієнтований на combined log format, який поширений у практиці Apache та Nginx. Цей компонент повинен не лише виділяти поля, а й відокремлювати некоректні рядки, які не відповідають очікуваній структурі. У результаті роботи парсера система отримує список подій, придатних для подальшої обробки. Важливо, щоб некоректні записи не призводили до зупинки всієї системи, адже в реальних логах можуть траплятися пошкоджені або нестандартні рядки. Тому більш стійким є підхід, за якого система аналізує всі коректні записи, а помилки парсингу фіксує окремо.

Компонент нормалізації забезпечує приведення даних до єдиного вигляду. URL може декодуватися, HTTP-метод переводитися до одного регістру, код відповіді — до числового формату, а час — до стандартного типу дати й часу. Це має важливе значення для коректної роботи алгоритму оцінювання ризику, оскільки підозрілі запити часто можуть бути записані в закодованому або зміненому вигляді. Наприклад, одна і та сама спроба path traversal може бути представлена різними варіантами запису. Якщо система не нормалізує URL, вона може не розпізнати такі запити як однакові за суттю. Таким чином, нормалізація є необхідною умовою якісного аналізу.

Компонент виявлення підозрілих ознак працює на основі набору правил. До таких правил можуть входити перевірка URL на наявність фрагментів, характерних для SQL-ін’єкцій, XSS, path traversal або звернення до службових файлів; перевірка HTTP-кодів відповідей; оцінювання кількості запитів з однієї IP-адреси; виявлення нетипових HTTP-методів; аналіз User-Agent. У цьому контексті доцільним є rule-based підхід, оскільки він є прозорим і добре підходить для дипломної реалізації. Адміністратор може бачити, яке саме правило

спрацювало, а отже система не є непрозорою для користувача. OWASP ASVS розглядає security logging and error handling як окремий напрям перевірки безпечності вебзастосунків, що підтверджує важливість фіксації та подальшого аналізу подій безпеки [49].

У практичній реалізації основний фрагмент логіки можна подати у вигляді узагальненої функції, яка приймає подію, перевіряє її на підозрілі ознаки та повертає рівень ризику:

```
    def calculate_risk(event, request_count):
score = 0
reasons = []
patterns = ["union", "select", "<script", "../", ".env", "admin", "wp-login"]
if any(pattern in event["url"].lower() for pattern in patterns):
    score += 2
    reasons.append("підозрілий шаблон у URL")
if event["status"] in [401, 403, 404]:
    score += 1
    reasons.append("помилка доступу або неіснуючий ресурс")
if event["method"] not in ["GET", "POST", "HEAD"]:
    score += 2
    reasons.append("нетиповий HTTP-метод")
if request_count > 100:
    score += 3
    reasons.append("надмірна кількість запитів з IP-адреси")
level = "high" if score >= 6 else "medium" if score >= 3 else "low"
return score, level, reasons
```

Наведений фрагмент не охоплює всю систему, але демонструє принцип її роботи. Кожна ознака додає певну кількість балів до загальної оцінки, після чого система визначає рівень ризику. Важливо, що функція повертає також пояснення

причин спрацювання. Це має практичне значення, оскільки адміністратор повинен бачити не лише рівень ризику, а й логіку, за якою система дійшла відповідного висновку. Такий підхід є більш обґрунтованим, ніж проста автоматична позначка небезпеки без пояснення.

Компонент збереження результатів відповідає за запис оброблених подій до бази даних. Для кожного запису доцільно зберігати вихідні поля логу, результати нормалізації, кількість балів ризику, рівень ризику та перелік причин спрацювання. Це дозволяє адміністративній панелі працювати не з сирими файлами, а з уже підготовленими структурованими даними. Крім того, збереження результатів дає змогу виконувати повторний аналіз, будувати статистику, переглядати історію активності, порівнювати різні періоди та використовувати систему для подальшого тестування.

Компонент адміністративної панелі забезпечує взаємодію користувача із системою. Його завдання полягає у відображенні загальної кількості оброблених записів, кількості підозрілих подій, розподілу за рівнями ризику, найбільш активних IP-адрес, найбільш частих кодів відповідей і списку подій із можливістю фільтрації. Доцільно передбачити фільтри за датою, IP-адресою, HTTP-методом, кодом відповіді та рівнем ризику. Окреме значення має сторінка детального перегляду події, де адміністратор може побачити вихідний запис, розпізнані поля та причини, через які система визнала подію підозрілою.

Наукове обґрунтування візуалізації результатів пов’язане з тим, що аналіз великих обсягів логів потребує не лише машинної обробки, а й зручного подання результатів людині. У дослідженнях observability підкреслюється, що сучасні програмні системи генерують значні обсяги телеметричних даних, і без належної агрегації та візуального представлення їх складно ефективно використовувати для прийняття рішень. Для цієї системи це означає, що таблиця логів має бути доповнена статистичними картками, графіками або діаграмами, які показують кількість подій за рівнем ризику, динаміку запитів у часі, частку помилок 4xx і 5xx та активність окремих IP-адрес.

Окремо потрібно врахувати питання безпечного виведення даних у вебінтерфейсі. Оскільки URL або User-Agent можуть містити шкідливі фрагменти, наприклад XSS-код, система не повинна виводити їх у HTML-сторінку без екранування. Якщо цього не зробити, адміністратор може відкрити сторінку з логами й ненавмисно виконати небезпечний фрагмент коду, який був записаний у журналі. Саме тому реалізація інтерфейсу повинна передбачати безпечне відображення значень, які надходять із лог-файлу. Це також узгоджується із загальними рекомендаціями OWASP щодо безпечного логування, обробки помилок і недопущення витоку або небезпечного відображення даних [49].

Загальну архітектуру програмної реалізації можна описати як послідовну взаємодію кількох модулів. Модуль завантаження приймає файл і передає його модулю парсингу. Модуль парсингу виділяє структуровані поля. Модуль нормалізації приводить їх до єдиного формату. Модуль аналізу застосовує правила виявлення підозрілих ознак. Модуль оцінювання ризику формує кількість балів і рівень небезпеки. Модуль збереження записує результати в базу даних. Модуль адміністративного інтерфейсу надає користувачу засоби перегляду, фільтрації та аналізу результатів. Така структура є логічною, оскільки кожний компонент виконує окрему роль і може бути змінений або вдосконалений без повної перебудови системи.

Отже, вибір технологій для реалізації системи аналізу логів вебсервера є обґрунтованим як з практичного, так і з наукового погляду. Python забезпечує зручну обробку текстових даних і реалізацію алгоритмів аналізу, Flask або подібний вебфреймворк дозволяє створити адміністративну панель, SQLite забезпечує просте локальне збереження результатів, а модульна архітектура робить систему зрозумілою, тестованою та придатною для подальшого розвитку. У науковому аспекті така реалізація відповідає підходам до log-based monitoring, observability, безпечного логування та побудови систем підтримки прийняття рішень у сфері кібербезпеки. Реалізовані функціональні компоненти створюють основу для подальшого тестування системи на змодельованих сценаріях потенційних кібератак і оцінювання її ефективності.

### 3.2. Реалізація адміністративної панелі, журналу подій та механізмів візуалізації результатів

Реалізація адміністративної панелі є одним із ключових етапів програмної реалізації системи аналізу логів вебсервера, оскільки саме через цей інтерфейс користувач отримує доступ до результатів обробки журналів подій, переглядає виявлені підозрілі записи, аналізує статистику та приймає рішення щодо подальшої перевірки потенційних інцидентів. Якщо алгоритм обробки логів виконує технічну частину роботи, пов’язану з парсингом, нормалізацією та оцінюванням ризику, то адміністративна панель перетворює ці результати на зрозумілу для користувача інформацію. Саме тому її реалізація повинна враховувати не лише програмну логіку, а й принципи зручності, структурованості, безпечного відображення даних і швидкого доступу до найбільш важливих подій.

Адміністративна панель у межах розроблюваної системи виконує роль центрального інтерфейсу взаємодії адміністратора з результатами аналізу логів. Її основне призначення полягає у тому, щоб користувач міг завантажити або переглянути оброблений лог-файл, побачити загальну статистику, оцінити кількість подій різного рівня ризику, відфільтрувати записи за потрібними параметрами та відкрити детальну інформацію про конкретну подію. Такий підхід є більш ефективним, ніж перегляд сирого текстового файлу, оскільки адміністратор отримує не хаотичний набір рядків, а впорядковану систему даних, де кожний запис має визначені поля, рівень ризику та пояснення причин спрацювання.

Загальна структура адміністративної панелі може складатися з кількох основних блоків: сторінки завантаження лог-файлу, головної інформаційної панелі, журналу подій, сторінки детального перегляду запису, блоку фільтрації та пошуку, а також розділу статистики й візуалізації. Така структура є логічною, оскільки вона відображає повний цикл роботи користувача із системою: спочатку адміністратор завантажує файл, потім система обробляє його, після цього

результати відображаються у вигляді таблиці й графічних показників, а за потреби користувач переходить до детального аналізу окремих подій.

На головній сторінці адміністративної панелі доцільно розмістити узагальнені показники, які дають швидке уявлення про стан проаналізованих логів. До таких показників належать загальна кількість оброблених записів, кількість записів із низьким, середнім і високим рівнем ризику, кількість унікальних IP-адрес, кількість помилок 4xx і 5xx, кількість підозрілих URL, а також час останнього аналізу. Такі інформаційні блоки дозволяють адміністратору одразу оцінити, чи містить журнал значну кількість підозрілих подій, і визначити, чи потрібно переходити до детального аналізу.

Журнал подій є основним елементом адміністративної панелі. Він повинен відображатися у вигляді таблиці, де кожний рядок відповідає одному запису логу або одній обробленій події. Доцільно передбачити такі колонки: дата й час запиту, IP-адреса, HTTP-метод, URL, код відповіді, User-Agent, кількість балів ризику, рівень ризику та коротке пояснення спрацювання. Такий формат дає можливість швидко переглядати події та порівнювати їх між собою. Наприклад, якщо в таблиці видно багато записів із однієї IP-адреси, що мають код 404 і звертаються до різних неіснуючих сторінок, адміністратор може зробити попередній висновок про можливе сканування вебресурсу.

Важливою функцією журналу подій є сортування. Користувач повинен мати можливість відсортувати записи за часом, рівнем ризику, IP-адресою, кодом відповіді або кількістю балів. Найбільш доцільним варіантом за замовчуванням є сортування за рівнем ризику та часом, щоб події з високим рівнем небезпеки відображалися у верхній частині таблиці. Це дозволяє адміністратору не витрачати час на перегляд усіх записів поспіль, а одразу перейти до найбільш важливих випадків.

Не менш важливою є фільтрація подій. У системі доцільно реалізувати фільтри за рівнем ризику, IP-адресою, HTTP-методом, кодом відповіді, часовим проміжком і частиною URL. Наприклад, адміністратор може обрати лише події з високим ризиком або переглянути всі запити, що завершилися кодом 403. Також

корисною є можливість фільтрації за конкретною IP-адресою, оскільки це дозволяє швидко проаналізувати поведінку окремого джерела запитів. Якщо з однієї адреси надходили часті звернення до адміністративних сторінок або неіснуючих ресурсів, фільтрація дасть змогу побачити повну картину активності цього клієнта.

Окрема сторінка детального перегляду події повинна містити розширену інформацію про конкретний запис. На ній доцільно відображати вихідний рядок логу, розпізнані поля, нормалізовані значення, рівень ризику, кількість балів і повний перелік причин, через які система позначила подію як підозрілу. Такий підхід є важливим, оскільки адміністратор повинен мати можливість перевірити не лише результат роботи алгоритму, а й логіку його формування. Наприклад, якщо подія отримала середній рівень ризику, користувач має бачити, що це сталося через звернення до адміністративної сторінки та код відповіді 403.

Реалізація журналу подій повинна враховувати проблему великого обсягу даних. Навіть у межах навчальної системи лог-файл може містити сотні або тисячі записів, тому виведення всіх подій на одній сторінці є незручним. Доцільно реалізувати пагінацію, тобто поділ таблиці на сторінки. Наприклад, на одній сторінці може відображатися 25, 50 або 100 записів. Це покращує зручність роботи з інтерфейсом і зменшує навантаження на браузер. У разі подальшого розвитку системи можна також реалізувати асинхронне завантаження даних, однак для дипломної роботи достатньо класичної пагінації.

Оскільки адміністративна панель відображає дані, отримані з лог-файлу, важливо забезпечити безпечне виведення інформації. URL, User-Agent або інші поля можуть містити фрагменти потенційно небезпечного коду. Наприклад, у логах можуть бути зафіксовані спроби XSS-атаки, і якщо система виведе такий фрагмент у HTML-сторінку без екранування, він може виконатися в браузері адміністратора. Тому всі значення, які відображаються у вебінтерфейсі, повинні екрануватися. Flask використовує шаблонізатор Jinja2, який підтримує автоматичне екранування для HTML-шаблонів, що є важливим для безпечного відображення даних, які походять із зовнішніх джерел. Це особливо актуально

саме для системи аналізу логів, оскільки журнал подій може містити фрагменти шкідливих запитів, які не повинні виконуватися під час перегляду в адміністративній панелі.

Механізм візуалізації результатів є необхідним доповненням до таблиці подій. Табличний формат зручний для детального аналізу окремих записів, однак він не завжди дозволяє швидко оцінити загальні закономірності. Саме тому доцільно додати графіки та діаграми, які відображають розподіл подій за рівнями ризику, кількість запитів у часі, частку HTTP-кодів відповідей, найактивніші IP-адреси та кількість підозрілих подій за категоріями. Візуальне представлення допомагає адміністратору швидше помітити аномальні піки, збільшення кількості помилок або концентрацію підозрілої активності з окремих джерел.

Для реалізації візуалізації можна використати бібліотеку Chart.js, оскільки вона дозволяє створювати лінійні графіки, стовпчикові діаграми, кругові діаграми та інші візуальні елементи без надмірної складності. У контексті цієї системи найбільш доречними є кілька типів графіків. Лінійний графік може показувати динаміку кількості запитів у часі. Стовпчикова діаграма може відображати найактивніші IP-адреси або найбільш часті коди відповідей. Кругова або кільцева діаграма може демонструвати співвідношення подій із низьким, середнім і високим рівнем ризику. Chart.js підтримує різні типи діаграм і може бути інтегрована у звичайну HTML-сторінку через JavaScript, що робить її придатною для реалізації адміністративної панелі в межах дипломної роботи.

Візуалізація рівнів ризику є одним із найважливіших елементів панелі. Наприклад, якщо більшість подій має низький рівень ризику, а високоризикових записів лише декілька, адміністратор може зосередитися саме на них. Якщо ж кількість подій із високим ризиком значна, це може свідчити про активну фазу підозрілої діяльності. Візуальне представлення такого співвідношення дозволяє швидше оцінити загальний стан безпеки вебресурсу, ніж перегляд великої таблиці записів.

Окремо доцільно реалізувати графік динаміки запитів у часі. Для цього всі події групуються за часовими інтервалами, наприклад за хвилинами або годинами.

Після цього система підраховує кількість запитів у кожному інтервалі та передає ці дані до графіка. Якщо на графіку видно різкий пік активності, це може бути ознакою сканування, brute force-атаки або навантажувальної активності. Такий графік є корисним не лише для виявлення атак, а й для загального аналізу роботи вебресурсу, оскільки він показує періоди найбільшого навантаження.

Ще одним корисним елементом є візуалізація HTTP-кодів відповідей. Наприклад, велика частка 404 може свідчити про сканування неіснуючих сторінок, значна кількість 403 — про спроби доступу до заборонених ресурсів, а збільшення 5xx — про проблеми на стороні сервера або перевантаження. У таблиці ці коди можуть бути розкидані серед великої кількості записів, тоді як діаграма одразу показує їх загальне співвідношення. Це допомагає адміністратору швидко зрозуміти, які типи відповідей переважають у проаналізованому журналі.

Для аналізу джерел активності важливим є блок найактивніших IP-адрес. У ньому можна відображати IP-адреси, з яких надійшла найбільша кількість запитів, а також кількість підозрілих подій для кожної з них. Такий блок дозволяє швидко виявити адреси, які потенційно пов’язані зі скануванням, автоматизованими запитами або спробами несанкціонованого доступу. Водночас сама велика кількість запитів не повинна автоматично розглядатися як доказ атаки, оскільки активність може належати легітимному користувачу, пошуковому роботу або системі моніторингу. Саме тому візуалізація має доповнюватися детальним переглядом подій.

Журнал подій також повинен мати механізм позначення критичності записів. У практичному інтерфейсі це може бути реалізовано через текстові мітки низький, середній і високий ризик, а також через візуальне виділення рядків. Проте важливо не перевантажувати інтерфейс надмірною кількістю кольорів або попереджень. Основна мета полягає не в тому, щоб зробити таблицю візуально складною, а в тому, щоб адміністратор швидко бачив, які записи потребують уваги. Тому найкращим варіантом є стримане виділення подій за рівнем ризику та можливість фільтрації за цим параметром.

Важливим компонентом адміністративної панелі є журнал спрацювань алгоритму. На відміну від звичайного журналу подій, він фіксує не всі записи, а лише ті випадки, коли система виявила певну підозрілу ознаку. У такому журналі доцільно зберігати час спрацювання, IP-адресу, URL, рівень ризику та причину. Наприклад, причиною може бути підозрілий шаблон у URL, часті помилки доступу, нетиповий HTTP-метод або перевищення порогу запитів. Журнал спрацювань є важливим для подальшого тестування системи, оскільки дозволяє перевірити, чи правильно алгоритм реагує на змодельовані сценарії атак.

Для реалізації журналу подій і панелі візуалізації база даних повинна містити достатньо структуровані записи. Доцільно передбачити поля для вихідних параметрів логу, нормалізованих значень, рівня ризику та причин спрацювання. Наприклад, у таблиці подій можуть бути такі поля: id, ip_address, request_time, method, url, status_code, user_agent, risk_score, risk_level, risk_reasons. Така структура дає можливість швидко отримувати дані для таблиці, фільтрів і графіків. Для побудови статистики можна виконувати групування за risk_level, status_code, ip_address або часовими інтервалами.

У межах адміністративної панелі також доцільно передбачити простий механізм пошуку. Користувач може вводити IP-адресу, частину URL або код відповіді, після чого система відображає лише ті події, які відповідають запиту. Такий механізм особливо корисний, якщо адміністратор уже має певну підозрілу IP-адресу або хоче перевірити всі звернення до конкретної сторінки. Пошук не повинен замінювати фільтрацію, але має доповнювати її, забезпечуючи швидкий доступ до потрібної інформації.

З технічного погляду реалізація адміністративної панелі може бути побудована за класичною схемою: серверна частина формує дані з бази, передає їх у HTML-шаблон, а клієнтська частина відображає таблиці та графіки. Для невеликої дипломної системи цього достатньо, оскільки основне навантаження пов’язане не з інтерактивністю, а з коректною обробкою і представленням результатів. У майбутньому систему можна розширити за рахунок API,

асинхронного оновлення таблиць, автоматичного періодичного аналізу нових логів або сповіщень про високоризикові події.

Приклад базової структури сторінки адміністративної панелі можна подати так:

```
        <div class="dashboard">
  <div class="card">Усього подій: {{ total_events }}</div>
  <div class="card">Високий ризик: {{ high_risk }}</div>
  <div class="card">Унікальні IP: {{ unique_ips }}</div>
</div>

<table>
  <tr>
    <th>Час</th>
    <th>IP</th>
    <th>Метод</th>
    <th>URL</th>
    <th>Код</th>
    <th>Ризик</th>
  </tr>
  {% for event in events %}
  <tr>
    <td>{{ event.request_time }}</td>
    <td>{{ event.ip_address }}</td>
    <td>{{ event.method }}</td>
    <td>{{ event.url }}</td>
    <td>{{ event.status_code }}</td>
    <td>{{ event.risk_level }}</td>
  </tr>
  {% endfor %}
</table>
```

Наведений фрагмент демонструє загальну ідею відображення статистичних карток і таблиці подій. У повній реалізації доцільно додати фільтри, пагінацію, посилання на детальний перегляд події та окремий блок графіків. Також потрібно врахувати безпечне відображення значень, які походять із лог-файлу, щоб уникнути виконання потенційно небезпечного вмісту.

Приклад передачі даних для діаграми ризиків може виглядати так:

```
<canvas id="riskChart"></canvas>
<script>
const ctx = document.getElementById('riskChart');
new Chart(ctx, {
  type: 'doughnut',
  data: {
    labels: ['Низький', 'Середній', 'Високий'],
    datasets: [{
      data: [{{ low_risk }}, {{ medium_risk }}, {{ high_risk }}]
    }]
  }
});
</script>
```

Цей приклад показує базову логіку побудови діаграми, яка відображає співвідношення подій за рівнями ризику. Для дипломної роботи такого фрагмента достатньо, оскільки він демонструє не повну промислову реалізацію, а принцип візуалізації результатів аналізу. Подібним способом можна побудувати графік кількості запитів у часі, діаграму HTTP-кодів або рейтинг активних IP-адрес.

Окремої уваги потребує питання зручності інтерфейсу. Адміністративна панель повинна бути зрозумілою навіть для користувача, який не переглядає серверні логи щодня. Назви колонок мають бути простими, рівні ризику — чіткими, а причини спрацювання — сформульованими зрозумілою мовою. Наприклад, замість технічного повідомлення pattern matched доцільно виводити виявлено підозрілий шаблон у URL. Це робить систему більш практичною,

оскільки її результати можуть використовувати не лише програмісти, а й адміністратори або фахівці з кібербезпеки початкового рівня.

У підсумку реалізація адміністративної панелі, журналу подій та механізмів візуалізації результатів забезпечує практичну цінність усієї системи аналізу логів вебсервера. Без такого інтерфейсу результати алгоритму залишалися б технічними даними, які складно швидко інтерпретувати. Адміністративна панель дозволяє перетворити ці дані на зручну інформаційну модель: користувач бачить загальну статистику, може переглядати детальні записи, фільтрувати події, аналізувати рівні ризику та оцінювати динаміку активності. Журнал подій забезпечує прозорість аналізу, а візуалізація допомагає швидко виявляти аномалії, піки навантаження та концентрацію підозрілої активності. Саме тому цей компонент є необхідною складовою програмної реалізації системи та створює основу для подальшого тестування її ефективності на змодельованих сценаріях потенційних кібератак.

### 3.3. Тестування системи на змодельованих сценаріях потенційних кібератак та оцінка її ефективності

Тестування системи аналізу логів вебсервера є завершальним етапом програмної реалізації, оскільки саме під час перевірки визначається, наскільки коректно система виконує поставлені завдання: обробляє журнали подій, виділяє структуровані поля, виявляє підозрілі ознаки, присвоює рівень ризику та відображає результати в адміністративній панелі. Якщо попередні підрозділи були спрямовані на обґрунтування вибору технологій, реалізацію функціональних компонентів і побудову інтерфейсу, то тестування дозволяє оцінити практичну працездатність розробленого рішення. У межах цієї роботи доцільно застосувати тестування на змодельованих сценаріях, оскільки воно дає змогу безпечно перевірити реакцію системи на типові прояви потенційних кібератак без впливу на реальний вебсервер, реальних користувачів або продуктивне середовище. Такий підхід узгоджується із загальною логікою технічного тестування інформаційної безпеки, де перевірка має бути плановану, контрольованою та спрямованою на оцінювання здатності системи виявляти ризикові події. NIST SP

800-115 розглядає тестування й оцінювання інформаційної безпеки як процес, що охоплює планування, виконання перевірок, аналіз результатів і формування висновків щодо стану захисту.

Основною метою тестування в межах цієї роботи є перевірка того, чи здатна система коректно класифікувати записи логів за рівнем ризику. Для цього було сформовано набір умовних логів, які імітують як нормальну активність користувачів, так і потенційно небезпечні сценарії. До нормальної активності належать звичайні GET-запити до головної сторінки, перегляд інформаційних розділів сайту, завантаження статичних ресурсів, отримання відповідей із кодами 200 або 302. До підозрілої активності віднесено звернення до адміністративних сторінок, часті помилки 404, спроби доступу до службових файлів, URL із фрагментами, характерними для SQL-ін’єкцій або XSS, нетипові HTTP-методи, повторювані POST-запити до сторінки авторизації та надмірну кількість запитів з однієї IP-адреси. Такий набір сценаріїв дозволяє перевірити не лише окремі правила, а й загальну здатність алгоритму оцінювати сукупність ознак.

Тестування системи доцільно організувати у кілька етапів. На першому етапі перевіряється правильність завантаження та зчитування лог-файлу. Система повинна приймати файл у визначеному форматі, відхиляти некоректні або порожні файли, а також не припиняти роботу в разі наявності окремих пошкоджених рядків. На другому етапі оцінюється коректність парсингу, тобто правильне виділення IP-адреси, часу запиту, HTTP-методу, URL, коду відповіді та User-Agent. На третьому етапі перевіряється нормалізація даних, зокрема приведення URL до єдиного вигляду, обробка регістру символів, перетворення кодів відповідей і часу. На четвертому етапі тестується робота правил виявлення підозрілої активності. На п’ятому етапі перевіряється формування рівня ризику та пояснення причин спрацювання. Завершальним етапом є перевірка адміністративної панелі, фільтрів, таблиці подій і графіків.

Для тестування було змодельовано кілька типових сценаріїв. Перший сценарій відображає нормальну поведінку користувача, який відкриває головну сторінку, переходить до кількох розділів сайту та отримує успішні відповіді

сервера. У такому випадку система повинна присвоїти подіям низький рівень ризику, оскільки відсутні підозрілі URL, помилки доступу, надмірна частота запитів або нетипові методи. Цей сценарій є важливим для перевірки того, що система не створює зайвих хибних спрацювань у випадку звичайної активності. Надмірна кількість хибних попереджень знижує практичну цінність будь-якої системи моніторингу, оскільки адміністратор поступово перестає сприймати попередження як значущі.

Другий сценарій імітує сканування директорій. У змодельованому журналі одна IP-адреса протягом короткого проміжку часу надсилає запити до великої кількості неіснуючих або службових ресурсів, зокрема до сторінок admin, backup, config, test, phpinfo, old, wp-login. Більшість таких запитів завершується кодами 403 або 404. Очікуваним результатом є присвоєння середнього або високого рівня ризику залежно від кількості запитів і характеру URL. Такий сценарій перевіряє здатність системи аналізувати не лише окремий запис, а й повторювану поведінку з однієї IP-адреси. У практичному аспекті саме сканування часто передує спробам експлуатації вразливостей, тому його своєчасне виявлення є важливим для адміністратора.

Третій сценарій імітує спробу SQL-ін’єкції. У логах містяться запити з параметрами, у яких наявні фрагменти на зразок union, select, or 1=1 або інші підозрілі комбінації. Очікується, що система виявить відповідний шаблон у URL, додасть бали ризику та сформує пояснення про наявність підозрілого параметра. При цьому важливо, щоб система не блокувала або не позначала як критичні всі запити лише через наявність окремого слова, а оцінювала його у поєднанні з іншими ознаками: кодом відповіді, частотою повторення, IP-адресою та типом ресурсу. OWASP Web Security Testing Guide використовується як практичний орієнтир для тестування вебзастосунків і містить підходи до перевірки конфігурації, введення даних та інших аспектів безпеки вебсистем.

Четвертий сценарій стосується XSS-подібних запитів, коли у URL або параметрах містяться фрагменти, схожі на script, javascript, onerror або інші ознаки спроби впровадження клієнтського коду. Основне завдання тестування полягає не

лише в тому, щоб перевірити виявлення таких фрагментів, а й у тому, щоб переконатися в безпечному відображенні цих значень в адміністративній панелі. Якщо система виводить URL без екранування, потенційно небезпечний фрагмент може виконатися у браузері адміністратора. Тому позитивним результатом тесту є ситуація, коли система позначає подію як підозрілу, але відображає її без виконання будь-якого коду. Це демонструє, що система не лише аналізує шкідливі запити, а й безпечно працює з ними в інтерфейсі.

П’ятий сценарій імітує brute force-активність щодо сторінки авторизації. У журналі подій багаторазово повторюються POST-запити до сторінки login або auth з однієї IP-адреси або групи адрес. Частина запитів завершується кодами 401 або 403, що може свідчити про невдалі спроби входу. Очікуваним результатом є підвищення рівня ризику для таких подій, оскільки система повинна враховувати не тільки URL сторінки авторизації, а й частоту повторення запитів. У цьому випадку важливо, щоб алгоритм не трактував один невдалий вхід як високу загрозу, але реагував на масову повторювану активність. Саме це демонструє перевагу risk-score підходу, який оцінює сукупність ознак, а не один ізольований параметр.

Шостий сценарій моделює HTTP flood або надмірну кількість запитів з однієї IP-адреси за короткий проміжок часу. У цьому випадку URL може бути звичайним, наприклад головна сторінка або популярний ресурс, однак підозрілим є не зміст запиту, а його частота. Система повинна визначити, що кількість запитів перевищує встановлений поріг, і підвищити рівень ризику. Такий сценарій є важливим, оскільки не всі атаки мають очевидні шкідливі параметри в URL. Частина небезпечної активності проявляється саме через інтенсивність, повторюваність і нетипову динаміку звернень.

Сьомий сценарій пов’язаний із використанням нетипових HTTP-методів. У змодельованих логах присутні запити з методами PUT, DELETE, OPTIONS або PATCH до ресурсів, де такі методи не передбачені логікою звичайної роботи вебсайту. Очікується, що система позначить такі події як потенційно підозрілі та додасть відповідну причину до пояснення ризику. Однак і тут важливо

враховувати контекст: для API-сервісів частина таких методів може бути нормальною, тоді як для простого інформаційного вебсайту вони можуть виглядати нетипово. Саме тому під час оцінки ефективності потрібно зазначити, що порогові значення й правила системи мають адаптуватися до конкретного вебресурсу.

Для фіксації результатів тестування доцільно використати таблицю, яка показує очікувану та фактичну реакцію системи на кожний сценарій.

| № | Сценарій тестування | Основні ознаки в логах | Очікувана реакція системи | Фактичний результат |
|---|---|---|---|---|
| 1 | Нормальна активність користувача | GET-запити, коди 200, звичайні URL | Низький ризик | Події класифіковано як низькоризикові |
| 2 | Сканування директорій | Багато URL, коди 403 і 404, службові шляхи | Середній або високий ризик | Виявлено підозрілу активність |
| 3 | SQL-ін’єкція | Параметри union, select, or 1=1 | Середній або високий ризик | Події позначено як підозрілі |
| 4 | XSS-подібний запит | script, javascript, onerror у URL | Середній ризик і безпечне відображення | Події виявлено, код не виконується |
| 5 | Brute force | Часті POST-запити до login, коди 401/403 | Високий ризик за повторюваності | Рівень ризику підвищено |
| 6 | HTTP flood | Надмірна кількість запитів за короткий час | Високий ризик | Активність позначено як небезпечну |
| 7 | Нетипові HTTP-методи | PUT, DELETE, OPTIONS, PATCH | Середній ризик | Події позначено як нетипові |

Оцінювання ефективності системи можна здійснювати за кількома показниками. Першим показником є коректність парсингу, тобто частка рядків журналу, які система змогла правильно розпізнати. Якщо більшість рядків відповідає типовому формату access log, очікуваним результатом є високий рівень коректного розпізнавання. Другим показником є точність класифікації ризику, тобто відповідність фактичного рівня ризику очікуваному результату для змодельованого сценарію. Третім показником є кількість хибних спрацювань, коли

нормальна активність помилково позначається як підозріла. Четвертим показником є кількість пропущених підозрілих подій, коли система не виявляє змодельований ризиковий сценарій. П’ятим показником є зручність інтерфейсу, зокрема можливість швидко знайти події з високим ризиком, переглянути причини спрацювання та оцінити загальну статистику.

Для більш формалізованої оцінки можна використати базові метрики: кількість правильно виявлених підозрілих подій, кількість правильно визначених звичайних подій, кількість хибних спрацювань і кількість пропущених ризикових подій. На основі цього можна розрахувати умовну точність системи в межах змодельованого набору даних. Наприклад, якщо було підготовлено 100 записів, серед яких 40 містили підозрілі ознаки, а система правильно виявила 36 із них і помилково позначила 5 нормальних подій як підозрілі, то можна зробити висновок про загалом прийнятну ефективність для базової rule-based системи. При цьому важливо наголосити, що такі результати мають умовний характер, оскільки отримані на змодельованому наборі даних, а не на реальному промисловому трафіку.

У межах тестування важливо оцінити не тільки точність виявлення, а й пояснюваність результатів. Система вважається ефективнішою, якщо вона не просто присвоює рівень ризику, а й показує, чому саме подія була позначена як підозріла. Наприклад, у журналі подій має відображатися, що ризик підвищено через підозрілий шаблон у URL, часті помилки 404, звернення до адміністративної сторінки або надмірну кількість запитів. Такий підхід відповідає практичній логіці безпекового моніторингу, оскільки адміністратор повинен мати змогу швидко перевірити аргументацію системи, а не працювати з непрозорими повідомленнями. OWASP Logging Cheat Sheet підкреслює значення логування для виявлення подій безпеки, розслідування інцидентів і підтримки моніторингу, що підтверджує важливість збереження змістовних пояснень до кожного спрацювання.

Окремо під час тестування перевіряється робота адміністративної панелі. Вона повинна коректно відображати загальну кількість подій, кількість записів за

рівнями ризику, таблицю журналу подій, фільтри, пошук і графіки. Якщо система правильно класифікує події, але інтерфейс не дозволяє зручно їх переглянути, практична цінність рішення знижується. Тому ефективність адміністративної панелі оцінюється за тим, чи може користувач швидко знайти високоризикові записи, переглянути активність окремої IP-адреси, відфільтрувати події за кодом відповіді та оцінити загальну динаміку запитів. У межах дипломної роботи достатньо підтвердити, що ці функції працюють на тестовому наборі даних.

Важливою частиною оцінювання є перевірка механізмів візуалізації. Побудовані графіки повинні відповідати даним, що зберігаються в базі. Наприклад, якщо в тестовому наборі є 10 подій із високим ризиком, діаграма ризиків повинна показувати саме цю кількість. Якщо в журналі змодельовано різкий пік запитів у певний часовий інтервал, графік динаміки повинен відобразити це зростання. Якщо переважають відповіді 404, відповідна діаграма HTTP-кодів повинна демонструвати цю закономірність. Таким чином, тестування візуалізації підтверджує, що система не лише правильно аналізує записи, а й коректно представляє результати користувачу.

За результатами тестування можна зробити висновок, що розроблена система є ефективною для базового виявлення потенційно підозрілої активності в логах вебсервера. Вона коректно обробляє типові записи access log, виділяє ключові поля, визначає підозрілі URL, реагує на помилки доступу, враховує надмірну частоту запитів, присвоює рівень ризику та пояснює причини спрацювання. Найкраще система виявляє ті сценарії, які мають чіткі формальні ознаки: сканування директорій, SQL-подібні параметри, XSS-подібні фрагменти, повторювані помилки доступу, нетипові HTTP-методи та надмірну кількість запитів. Водночас складніші атаки, які не залишають очевидних слідів у access logs, можуть потребувати додаткових джерел даних, наприклад журналів застосунку, WAF, систем автентифікації або мережевого моніторингу.

До обмежень розробленої системи слід віднести залежність від якості вхідних логів, обмеженість набору правил, можливість хибних спрацювань і потребу в адаптації порогових значень до конкретного вебресурсу. Наприклад,

однаковий рівень активності може бути нормальним для великого вебсервісу, але підозрілим для невеликого сайту. Також система не може достовірно визначити намір користувача лише за одним записом журналу. Її завдання полягає не в остаточному підтвердженні факту атаки, а в автоматизованому виявленні потенційно ризикових подій, які потребують уваги адміністратора.

Перспективами подальшого вдосконалення системи є розширення набору правил, підтримка кількох форматів логів, інтеграція з журналами вебзастосунку, використання зовнішніх індикаторів компрометації, додавання сповіщень про високоризикові події, застосування машинного навчання для виявлення нетипових поведінкових моделей, а також підключення до SIEM або WAF. NIST SP 800-92 наголошує, що ефективне управління журналами передбачає не лише збереження подій, а й їх аналіз, захист і використання для реагування на інциденти, тому подальший розвиток системи може бути спрямований саме на посилення інтеграції з ширшою інфраструктурою кібербезпеки.

Отже, тестування на змодельованих сценаріях показало, що розроблена система виконує основні функції, визначені на етапі проєктування. Вона забезпечує автоматизовану обробку логів вебсервера, виявляє типові ознаки потенційних кібератак, формує рівень ризику, надає пояснення спрацювань і відображає результати в адміністративній панелі. Отримані результати підтверджують практичну придатність системи для первинного моніторингу вебсерверних логів і створюють основу для її подальшого розвитку як більш комплексного інструменту кібербезпеки.

**Висновки до розділу 3**

У третьому розділі було розглянуто програмну реалізацію та тестування системи аналізу логів вебсервера для виявлення потенційних кібератак. На основі функціональних вимог і алгоритму, сформованих у другому розділі, було

обґрунтовано вибір технологій, описано реалізацію основних програмних компонентів, адміністративної панелі, журналу подій, механізмів візуалізації результатів, а також проведено перевірку працездатності системи на змодельованих сценаріях підозрілої активності.

У процесі реалізації системи було обґрунтовано доцільність використання технологічного підходу, який поєднує простоту, відтворюваність і достатню функціональність для виконання завдань дипломної роботи. Основними компонентами системи визначено модуль завантаження лог-файлу, модуль парсингу записів, модуль нормалізації даних, модуль виявлення підозрілих ознак, модуль оцінювання рівня ризику, базу даних для збереження результатів, адміністративну панель і блок візуалізації статистики. Такий поділ системи на окремі функціональні частини дозволив забезпечити логічну структуру програмного рішення, спростити його тестування та створити основу для подальшого розширення.

Встановлено, що реалізація системи аналізу логів повинна спиратися не лише на технічну обробку рядків журналу, а й на загальну логіку управління подіями безпеки. NIST SP 800-92 розглядає log management як процес, що охоплює збирання, зберігання, аналіз і захист журналів, а не лише їх пасивне накопичення, тому розроблена система була орієнтована саме на аналітичну обробку записів і формування практично корисних висновків для адміністратора.

У розділі було показано, що адміністративна панель є важливою складовою системи, оскільки вона забезпечує зручну взаємодію користувача з результатами аналізу. Через адміністративний інтерфейс користувач може переглядати оброблені події, фільтрувати записи за рівнем ризику, IP-адресою, HTTP-методом, кодом відповіді або URL, а також відкривати детальну інформацію про конкретну подію. Особливе значення має відображення причин спрацювання алгоритму, оскільки адміністратор повинен бачити не лише кінцевий рівень ризику, а й ті ознаки, які вплинули на його формування.

Окремо було розглянуто реалізацію журналу подій, який виступає основним інформаційним елементом системи. У журналі подій кожний запис містить

структуровані параметри: час запиту, IP-адресу, HTTP-метод, URL, код відповіді, User-Agent, кількість балів ризику, рівень ризику та пояснення спрацювання. Такий формат є значно ефективнішим, ніж ручний перегляд сирого лог-файлу, оскільки дозволяє швидко знаходити найбільш важливі події, аналізувати поведінку окремих IP-адрес і визначати повторювані сценарії підозрілої активності.

У процесі реалізації механізмів візуалізації було визначено, що таблиця подій повинна доповнюватися графічними елементами, які допомагають швидко оцінити загальну картину. До таких елементів належать діаграма розподілу подій за рівнями ризику, графік динаміки запитів у часі, діаграма HTTP-кодів відповідей і блок найактивніших IP-адрес. Візуалізація дає змогу швидше виявляти аномальні піки активності, значну кількість помилок доступу, зростання подій із високим ризиком або концентрацію підозрілої активності з окремих джерел.

Також було наголошено на необхідності безпечного відображення даних у адміністративній панелі. Оскільки серверні логи можуть містити підозрілі URL, XSS-подібні фрагменти або нетипові User-Agent, їх не можна виводити в інтерфейс без належного екранування. OWASP звертає увагу, що неправильне логування може призводити до витоку інформації, log injection, недостатнього моніторингу та послаблення реагування на інциденти, тому безпечна обробка й відображення журналів є важливою умовою практичної придатності такої системи.

У межах тестування системи було змодельовано кілька типових сценаріїв потенційних кібератак: нормальну активність користувача, сканування директорій, спроби SQL-ін’єкцій, XSS-подібні запити, brute force-активність, HTTP flood і використання нетипових HTTP-методів. Результати тестування показали, що система здатна коректно обробляти типові записи access log, виділяти основні параметри запиту, виявляти підозрілі ознаки, присвоювати відповідний рівень ризику та пояснювати причини спрацювання. Найбільш ефективно система проявила себе у сценаріях, де підозріла активність має чіткі формальні ознаки:

підозрілі шаблони в URL, часті коди 403 або 404, надмірна кількість запитів, звернення до адміністративних сторінок і використання нетипових HTTP-методів.

Оцінка ефективності показала, що запропонована система є придатною для первинного моніторингу логів вебсервера та виявлення потенційно небезпечної активності. Вона не замінює повноцінні SIEM, WAF або IDS/IPS-рішення, однак може виконувати роль допоміжного інструменту для адміністратора, особливо у навчальному, тестовому або невеликому прикладному середовищі. Її перевагами є простота реалізації, прозорість логіки, пояснюваність результатів, можливість налаштування правил і зручне представлення даних у адміністративній панелі.

Разом із тим було визначено й певні обмеження системи. Її ефективність залежить від якості вхідних логів, правильності налаштування формату журналювання, повноти записаних параметрів і коректності встановлених порогових значень. Система краще виявляє атаки з явними ознаками в access logs, але складніші сценарії можуть потребувати додаткових джерел інформації, зокрема журналів вебзастосунку, систем автентифікації, WAF, мережевого моніторингу або зовнішніх індикаторів компрометації. Також rule-based підхід може давати хибні спрацювання або пропускати нові варіанти атак, якщо вони не відповідають заданим правилам.

Отже, третій розділ підтвердив практичну можливість реалізації системи аналізу логів вебсервера для виявлення потенційних кібератак. Розроблена система забезпечує завантаження та обробку логів, структуризацію записів, оцінювання рівня ризику, збереження результатів, перегляд журналу подій, візуалізацію статистики та тестування на змодельованих сценаріях. Отримані результати свідчать, що запропоноване рішення може використовуватися як основа для подальшого вдосконалення, зокрема через інтеграцію з SIEM/WAF, розширення набору правил, підключення зовнішніх індикаторів компрометації, автоматичне сповіщення адміністратора та застосування методів машинного навчання для виявлення складніших поведінкових аномалій.

# ВИСНОВОКИ

У процесі виконання роботи було досліджено теоретичні, проєктні та практичні аспекти розробки системи аналізу логів вебсервера для виявлення потенційних кібератак. Актуальність обраної теми підтверджується тим, що вебсервери є одним із найбільш відкритих і водночас найбільш уразливих елементів інформаційної інфраструктури, оскільки саме через них здійснюється взаємодія користувачів, вебзастосунків, API-сервісів і зовнішніх клієнтів. У таких умовах журнали подій набувають особливого значення, адже вони фіксують фактичні звернення до ресурсу, помилки доступу, підозрілі URL, нетипові HTTP-методи, частоту запитів і поведінкові ознаки, які можуть свідчити про потенційні кібератаки. NIST розглядає управління журналами як процес, що охоплює збирання, зберігання, аналіз і захист логів, тому їх використання для моніторингу безпеки є обґрунтованим і практично важливим напрямом.

У першому розділі було розкрито теоретичні основи аналізу логів вебсервера в системі кібербезпеки. Визначено, що логи вебсервера є структурованим або напівструктурованим джерелом технічної інформації, яке містить дані про IP-адресу клієнта, час запиту, HTTP-метод, URL, код відповіді сервера, розмір відповіді, referer, User-Agent та інші параметри. Було встановлено, що кожне з цих полів має окреме аналітичне значення: IP-адреса дозволяє групувати активність за джерелом, часова мітка — оцінювати інтенсивність дій, URL — виявляти спроби доступу до службових або небезпечних ресурсів, а коди відповідей — визначати характер реакції сервера на запит.

Також у першому розділі було охарактеризовано основні види кібератак на вебсервери та їхні прояви в журналах подій. До таких загроз належать SQL-ін’єкції, XSS-подібні запити, path traversal, brute force, credential stuffing, сканування директорій, HTTP flood, спроби доступу до адміністративних сторінок, використання нетипових HTTP-методів і звернення до службових або конфігураційних файлів. Було доведено, що більшість таких атак залишає певні прямі або непрямі сліди в access logs, хоча сам факт наявності окремої ознаки ще не є абсолютним доказом атаки. Саме тому ефективний аналіз логів повинен

ґрунтуватися на комплексному зіставленні кількох параметрів, а не на ізольованому перегляді окремих записів.

У межах теоретичного аналізу було визначено основні методи обробки серверних логів: парсинг, нормалізацію, фільтрацію технічного шуму, сигнатурний аналіз, евристичне оцінювання, статистичний аналіз, поведінковий аналіз, кореляцію подій, використання індикаторів компрометації та візуалізацію результатів. Найбільш доцільним для меж цієї роботи було визначено поєднання rule-based підходу та risk-score моделі, оскільки такий підхід є прозорим, зрозумілим, придатним для пояснення результатів і не потребує великого набору навчальних даних, як методи машинного навчання.

У другому розділі було сформовано функціональні вимоги до системи аналізу логів вебсервера. До основних вимог віднесено можливість завантаження лог-файлу, перевірку його коректності, парсинг записів, нормалізацію даних, виявлення підозрілих URL, аналіз HTTP-кодів відповідей, оцінювання частоти запитів, формування рівня ризику, збереження результатів у базі даних, перегляд журналу подій, фільтрацію, пошук і візуалізацію статистики. Було встановлено, що система повинна бути не лише технічно працездатною, а й зручною для адміністратора, оскільки її головне завдання полягає в підтримці швидкого виявлення потенційно небезпечних подій.

У межах проєктування було розроблено алгоритм обробки логів та оцінювання рівня ризику. Алгоритм передбачає послідовне виконання кількох етапів: отримання лог-файлу, перевірку вхідних даних, парсинг рядків, нормалізацію полів, попередню класифікацію подій, пошук підозрілих шаблонів, аналіз HTTP-кодів, оцінювання частоти запитів, розрахунок балів ризику та визначення підсумкового рівня небезпеки. Така послідовність дозволяє перетворити неструктурований текстовий журнал на набір зрозумілих подій, кожна з яких має визначені параметри, рівень ризику та пояснення причин спрацювання.

Важливим результатом другого розділу стало обґрунтування бальної моделі ризику. У межах цієї моделі кожна підозріла ознака додає певну кількість балів до

загального показника. Наприклад, підозрілий URL, помилки доступу, нетиповий HTTP-метод, звернення до адміністративного ресурсу або надмірна кількість запитів можуть підвищувати рівень ризику події. Після підсумовування балів система класифікує подію як низькоризикову, середньоризикову або високоризикову. Перевагою такого підходу є пояснюваність результату, оскільки адміністратор бачить не лише підсумковий рівень ризику, а й конкретні причини, через які система позначила подію як підозрілу.

У третьому розділі було розглянуто програмну реалізацію системи. Обґрунтовано вибір технологій, які дозволяють реалізувати систему в межах дипломної роботи без надмірної складності. Було показано, що доцільним є використання мови Python для обробки логів, регулярних виразів для парсингу, локальної бази даних для збереження результатів, вебінтерфейсу для адміністративної панелі та бібліотек візуалізації для побудови графіків. Такий стек технологій забезпечує достатню функціональність, простоту тестування, зрозумілість коду та можливість подальшого розширення системи.

Реалізована система складається з кількох основних функціональних компонентів: модуля завантаження лог-файлу, модуля парсингу, модуля нормалізації, модуля виявлення підозрілих ознак, модуля оцінювання ризику, бази даних, журналу подій, адміністративної панелі та блоку візуалізації. Такий модульний підхід дозволяє логічно розділити функції системи, спростити її підтримку та забезпечити можливість подальшого вдосконалення окремих елементів без повної перебудови програмного рішення.

Особливу увагу було приділено адміністративній панелі, яка забезпечує практичну взаємодію користувача із системою. Через адміністративний інтерфейс можна переглядати оброблені події, сортувати їх за рівнем ризику, фільтрувати за IP-адресою, HTTP-методом, кодом відповіді або URL, а також відкривати детальну інформацію про причини спрацювання алгоритму. Було встановлено, що саме поєднання журналу подій і механізмів візуалізації робить систему практично корисною, оскільки адміністратор отримує не просто набір технічних записів, а зрозумілу інформаційну картину стану вебресурсу.

У роботі було враховано питання безпечного відображення даних у вебінтерфейсі. Оскільки логи можуть містити фрагменти потенційно небезпечних запитів, зокрема XSS-подібні URL або нестандартні User-Agent, система не повинна виводити такі значення без екранування. OWASP підкреслює, що неправильна практика логування може спричиняти витік інформації, log injection, недостатній моніторинг і послаблення реагування на інциденти, тому безпечна обробка журналів є важливою умовою функціонування подібної системи.

Під час тестування системи було використано змодельовані сценарії потенційних кібератак. Зокрема, перевірялися сценарії нормальної активності користувача, сканування директорій, SQL-ін'єкцій, XSS-подібних запитів, brute force-активності, HTTP flood і використання нетипових HTTP-методів. Результати тестування показали, що система здатна коректно обробляти типові записи access log, виділяти ключові параметри, визначати підозрілі ознаки, присвоювати рівень ризику та відображати пояснення спрацювань у журналі подій.

Оцінка ефективності підтвердила, що розроблена система є придатною для первинного моніторингу логів вебсервера. Найбільш ефективно вона виявляє ті сценарії, які мають чіткі формальні ознаки: звернення до службових сторінок, підозрілі параметри в URL, часті відповіді 403 або 404, надмірну кількість запитів, повторювані POST-запити до сторінки авторизації та нетипові HTTP-методи. Водночас було визначено, що система не замінює повноцінні SIEM, WAF або IDS/IPS-рішення, а виступає як допоміжний інструмент для автоматизації первинного аналізу та швидкого виявлення потенційно ризикових подій.

У процесі роботи було досягнуто поставленої мети, яка полягала в розробці системи аналізу логів вебсервера для автоматизованого виявлення потенційних кібератак, оцінювання рівня ризику та відображення результатів моніторингу в зручному для адміністратора вигляді. Для досягнення цієї мети було досліджено сутність і структуру логів вебсервера, проаналізовано основні види кібератак і їх ознаки в журналах подій, визначено методи аналізу логів, сформовано функціональні вимоги, розроблено алгоритм оцінювання ризику, реалізовано

основні програмні компоненти та проведено тестування системи на змодельованих сценаріях.

Практичне значення роботи полягає в тому, що запропонована система може бути використана як навчальний або допоміжний інструмент для адміністраторів вебресурсів, студентів, розробників і фахівців початкового рівня у сфері кібербезпеки. Вона дозволяє автоматизувати первинну обробку серверних логів, зменшити обсяг ручної роботи, швидше виявляти підозрілі записи, оцінювати рівень ризику та формувати загальне уявлення про характер активності на вебсервері.

До обмежень розробленої системи слід віднести залежність від якості вхідних логів, обмеженість набору правил, можливість хибних спрацювань, необхідність налаштування порогових значень під конкретний вебресурс і складність виявлення атак, які не залишають очевидних ознак у access logs. У майбутньому систему доцільно вдосконалити шляхом підтримки кількох форматів логів, інтеграції з WAF або SIEM, підключення зовнішніх індикаторів компрометації, реалізації автоматичних сповіщень, розширення набору правил і застосування методів машинного навчання для виявлення складніших поведінкових аномалій.

Отже, виконана робота підтверджує, що аналіз логів вебсервера є важливим напрямом забезпечення кібербезпеки, а розробка автоматизованої системи для їх обробки дозволяє підвищити ефективність виявлення потенційних кібератак. Запропоноване рішення поєднує теоретично обґрунтовані підходи до log management, практичний алгоритм оцінювання ризику, програмну реалізацію основних функціональних компонентів і тестування на змодельованих сценаріях. Це дає підстави вважати розроблену систему придатною для використання як базового інструменту моніторингу вебсерверних логів і перспективною основою для подальшого розвитку більш комплексних засобів кібербезпеки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Apache Software Foundation. Log Files: Apache HTTP Server Version 2.4. URL: https://httpd.apache.org/docs/2.4/logs.html

NGINX. Access Logs: NGINX Documentation. URL: https://docs.nginx.com/waf/logging/access-logs/

Grace L. K. J., Maheswari V., Nagamalai D. Analysis of Web Logs and Web User in Web Mining. arXiv, 2011. URL: https://arxiv.org/abs/1101.5668

MDN Web Docs. HTTP Response Status Codes. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Mozilla Developer Network. HTTP Methods. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Zhang T., Qiu H., Castellano G., Rifai M., Chen C. S., Pianese F. System Log Parsing: A Survey. arXiv, 2022. URL: https://arxiv.org/abs/2212.14277

Гнатюк С., Побережна З., Скуратівський А. Метод інтегрування вимог кібербезпеки в життєвий цикл розроблення програмного забезпечення. Кібербезпека: освіта, наука, техніка. 2026. DOI: https://doi.org/10.28925/2663-4023.2026.32.1184 URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/1184

Супруненко І., Рудницький В. Хмарно-орієнтована архітектура імплементації методу адаптивного логування. Кібербезпека: освіта, наука, техніка. 2025. DOI: https://doi.org/10.28925/2663-4023.2025.27.724 URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/724

Педяш В., Ледовський Є., Ткач В. Сучасні методи виявлення шкідливого програмного забезпечення. Herald of Khmelnytskyi National University. Technical Sciences. 2024. № 333(2). С. 389–392. DOI: https://doi.org/10.31891/2307-5732-2024-333-2-60 URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/164

· OWASP Foundation. OWASP Top 10:2021. URL: https://owasp.org/Top10/2021/

Терещенко К., Терещенко Т., Черниш Ю., Штонда Р., Бокій О. Аналіз проблеми SQL-ін’єкцій в веб-застосунках. Кібербезпека: освіта, наука, техніка. 2024. № 1(25). С. 177–199. DOI: https://doi.org/10.28925/2663-4023.2024.25.177199 URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/615

Тишик І. Дослідження механізмів обходу фаєрволів веб-додатків. Кібербезпека: освіта, наука, техніка. 2025. № 3(31). С. 86–99. DOI: https://doi.org/10.28925/2663-4023.2025.31.1008 URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/1008

Маркевич М. Дослідження ефективності серверних атак на реляційні та нереляційні бази даних. Створення стратегії захисту. Кібербезпека: освіта, наука, техніка. 2025. URL: https://www.csecurity.kubg.edu.ua/index.php/journal/article/view/756

MITRE. CWE-22: Improper Limitation of a Pathname to a Restricted Directory. URL: https://cwe.mitre.org/data/definitions/22.html

Толкачова А., Піскозуб А. Методи для тестування безпеки веб-застосунків. Кібербезпека: освіта, наука, техніка. 2024. № 2(26). С. 115–122. DOI: https://doi.org/10.28925/2663-4023.2024.26.668 URL: https://www.csecurity.kubg.edu.ua/index.php/journal/article/view/668

Cloudflare. HTTP Flood DDoS Attack. URL: https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/

MITRE. CWE-89: Improper Neutralization of Special Elements used in an SQL Command. URL: https://cwe.mitre.org/data/definitions/89.html

MITRE. CWE-79: Improper Neutralization of Input During Web Page Generation. URL: https://cwe.mitre.org/data/definitions/79.html

OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html/

Zhang T., Qiu H., Castellano G., Rifai M., Chen C. S., Pianese F. System Log Parsing: A Survey. arXiv. 2022. URL: https://arxiv.org/abs/2212.14277

Apache Software Foundation. Log Files: Apache HTTP Server Version 2.4. URL: https://httpd.apache.org/docs/2.4/logs.html ; NGINX. Configuring Logging. URL: https://docs.nginx.com/nginx/admin-guide/monitoring/logging/

Педяш В., Ледовський Є., Ткач В. Сучасні методи виявлення шкідливого програмного забезпечення. Herald of Khmelnytskyi National University. Technical Sciences. 2024. № 333(2). С. 389–392. DOI: https://doi.org/10.31891/2307-5732-2024-333-2-60 URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/164

Бурячок В. Л., Складанний П. М., Хорошко В. О., Хохлачова Ю. Є., Хлапонін Ю. І. Аналіз методів, способів, механізмів, інструментів теорії прийняття рішень для моделювання систем захисту інформації. Кібербезпека: освіта, наука, техніка. 2022. № 4(16). С. 159–171. DOI: https://doi.org/10.28925/2663-4023.2022.16.159171 URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/373

OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Чернящук Н. Використання індикаторів компрометації для виявлення кібератак. Кібербезпека: освіта, наука, техніка. 2026. № 4(32). С. 8–19. DOI: https://doi.org/10.28925/2663-4023.2026.32.1034 URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/1034

Кравченко О., Кравченко В. Аналіз актуальних проблем безпеки корпоративних баз даних в умовах сучасної інфраструктури та шляхи їх вирішення. Кібербезпека: освіта, наука, техніка. 2025. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/726

Wijesinghe N., Hemmati H. Log-based Anomaly Detection of Enterprise Software: An Empirical Study. arXiv. 2023. URL: https://arxiv.org/abs/2310.20492

Xie Y., Zhang H., Babar M. A. LogGD: Detecting Anomalies from System Logs by Graph Neural Networks. arXiv. 2022. URL: https://arxiv.org/abs/2209.07869

OWASP Foundation. Poor Logging Practice. URL: https://owasp.org/www-community/vulnerabilities/Poor_Logging_Practice

OWASP Foundation. Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/

Apache Software Foundation. Log Files: Apache HTTP Server Version 2.4. URL: https://httpd.apache.org/docs/2.4/logs.html ; NGINX. Configuring Logging. URL: https://docs.nginx.com/nginx/admin-guide/monitoring/logging/

Kent K., Souppaya M. Guide to Computer Security Log Management: NIST Special Publication 800-92. Gaithersburg : National Institute of Standards and Technology, 2006. URL: https://csrc.nist.gov/pubs/sp/800/92/final

OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

OWASP Foundation. Web Security Testing Guide: Test Application Platform Configuration. URL: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/02-Test_Application_Platform_Configuration

Dempsey K., Chawla N., Johnson A., Johnston R., Jones A., Orebaugh A., Scholl M., Stine K. Information Security Continuous Monitoring for Federal Information Systems and Organizations: NIST Special Publication 800-137. Gaithersburg : National Institute of Standards and Technology, 2011. URL: https://csrc.nist.gov/pubs/sp/800/137/final

Kent K., Souppaya M. Guide to Computer Security Log Management: NIST Special Publication 800-92. Gaithersburg : National Institute of Standards and Technology, 2006. URL: https://csrc.nist.gov/pubs/sp/800/92/final

Apache Software Foundation. Log Files: Apache HTTP Server Version 2.4. URL: https://httpd.apache.org/docs/current/logs.html ; NGINX. Configuring Logging. URL: https://docs.nginx.com/nginx/admin-guide/monitoring/logging/

OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

NGINX. Access Logs. NGINX Documentation. URL: https://docs.nginx.com/waf/logging/access-logs/

OWASP Foundation. Poor Logging Practice. URL: https://owasp.org/www-community/vulnerabilities/Poor_Logging_Practice

Taher A., Bezemer C. P., Hassan A. E. Studying the Use of Logs in Software Monitoring: A Systematic Mapping Study. arXiv. 2019. URL: https://arxiv.org/abs/1912.05878

Python Software Foundation. Python 3 Documentation. URL: https://docs.python.org/3/

Розробка системи виявлення кіберзагроз на основі аналізу даних з веб-ресурсів на мові програмування Python. Системи управління, навігації та зв'язку. Збірник наукових праць. URL: https://journals.nupp.edu.ua/sunz/article/view/2565

SQLite Consortium. About SQLite. URL: https://www.sqlite.org/about.html

SQLAlchemy. SQLAlchemy ORM Documentation. URL: https://docs.sqlalchemy.org/20/orm/

Niedermaier S., Koetter F., Freymann A., Wagner S. Cloud-Native Observability: The Many-Faceted Benefits of Structured and Unified Logging — A Multi-Case Study. Future Internet. 2022. Vol. 14, No. 10. Article 274. URL: https://www.mdpi.com/1999-5903/14/10/274

OWASP Foundation. Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/

Hasselbring W., Henning S., Latte B., Möbius A., Richter T., Schalk S., Wojcieszak M. Industrial Survey on Microservice Tracing and Analysis. Empirical Software Engineering. 2021. URL: https://link.springer.com/article/10.1007/s10664-021-10063-9