ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА СИСТЕМ ТА ТЕХНОЛОГІЙ КІБЕРБЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Модуль захисту інформаційного ресурсу від SQL–ін'єкцій»

зі спеціальності *125 Кібербезпека*

(код, найменування спеціальності)

освітньо-професійної програми *Інформаційна та кібернетична безпека*

(назва програми)

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

_____ Рябий Денис

(підпис)

Виконав: здобувач вищої освіти групи БСД-43

Рябий Денис

(прізвище, ім'я)

Керівник д. т. н., професор, Зибін С.В.

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

Рецензент _____

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

Київ 2025

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

Кафедра Систем та технологій кібербезпеки
Ступінь вищої освіти Бакалавр
Спеціальність 125 Кібербезпека
Освітньо-професійна програма Інформаційна та кібернетична безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
_____ Галина ГАЙДУР
“___” _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Рябому Денису Олександрович
(прізвище, ім’я)

1. Тема кваліфікаційної роботи:

Модуль захисту інформаційного ресурсу від SQL–ін'єкцій

керівник кваліфікаційної роботи Зибін Сергій, д. т. н., професор
(прізвище, ім’я, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 2025 року № ____.

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 06.06.2025 р.

3. Вихідні дані до кваліфікаційної роботи
інформаційні ресурси організації;
рішення Web Application Firewall;
наукова та технічна література, експлуатаційна документація, нормативні документи, міжнародні стандарти.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження загроз безпеки інформаційних ресурсів веб-додатків .
2. Аналіз методів та засобів захисту від SQl-ін’єкцій.
3. Розроблення рекомендацій щодо захисту від SQl-ін’єкцій.

5. Перелік графічного матеріалу
Презентація PowerPoint.

6. Дата видачі завдання 28.02.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ зп	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Визначення актуальності проблеми захисту інформаційного ресурсу від SQl-ін’єкції	18.03.2025	
2.	Аналіз наукової та технічної літератури з питань теми кваліфікаційної роботи.	29.03.2025	
3.	Аналіз методів та засобів захисту від SQl-ін’єкції .	08.04.2025	
4.	Практична реалізація варіанту захисту від SQl-ін’єкції.	08.04.2025	
5.	Розроблення рекомендацій щодо застосування методів та засобів захисту від SQl-ін’єкції.	22.05.2025	
6.	Оформлення результатів дослідження.	03.06.2025	
7.	Підготовка доповіді до захисту.	06.06.2025 р.	

Здобувач вищої освіти _____ Денис РЯБИЙ
(підпис) *(ім’я, прізвище)*

Керівник кваліфікаційної роботи _____ Сергій ЗИБІН
(підпис) *(ім’я, прізвище)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу

здобувача РЯБОГО Дениса
на тему: «Модуль захисту інформаційного ресурсу від SQL–ін'єкцій»

Актуальність:

Позитивні сторони:

Недоліки:

Висновок:

Рецензент:

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

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

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на здобуття освітнього ступеня бакалавра

Направляється здобувач РЯБИЙ Денис до захисту кваліфікаційної роботи
(прізвище, ім'я)
спеціальності 125 Кібербезпека
освітньо-професійної програми Інформаційна та кібернетична безпека
(шифр і назва спеціальності)
на тему: «Модуль захисту інформаційного ресурсу від SQL–ін'єкцій».

Кваліфікаційна робота і рецензія додаються.

Директор інституту _____ Євгенія ІВАНЧЕНКО
(підпис) *(ім'я, прізвище)*

Висновок керівника кваліфікаційної роботи

Здобувач ***РЯБИЙ Денис*** обрав тему роботи, метою якої було дослідити методи та засоби захисту кінцевих точок організації та розробка рекомендацій щодо її реалізації. Перелік використаних джерел свідчить про вміння здобувачем розбиратись в наукових питаннях та застосовувати їх при дослідженнях. Під час виконання кваліфікаційної роботи ***РЯБИЙ Денис*** показав добру теоретичну та практичну підготовку, вміння самостійно вирішувати питання і робити висновки. Роботу виконував сумлінно, акуратно та вчасно за планом.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача ***РЯБИЙ Денис*** на оцінку **“”** та присвоїти йому кваліфікацію бакалавр з кібербезпеки за освітньою програмою Інформаційна та кібернетична безпека.

Керівник кваліфікаційної роботи _____ Сергій ЗИБІН
(підпис) *(ім'я, прізвище)*
“____” _____ 2025 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач **РЯБИЙ Денис** допускається до захисту даної кваліфікаційної роботи в Екзаменаційній комісії.

Завідувач кафедри Систем та технологій кібербезпеки
(назва)
_____ Галина ГАЙДУР
(підпис) *(ім'я, прізвище)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи: сторінки, рисунків, джерел.

Об'єкт дослідження – процес забезпечення захисту інформаційного ресурсу від SQL–ін'єкцій.

Предмет дослідження – методи та засоби захисту інформаційного ресурсу від SQL–ін'єкцій.

Мета роботи є застосунок системи захисту інформаційного ресурсу від SQL-ін'єкцій, реалізований за допомогою інструментів безпеки, таких як ModSecurity, та формування рекомендацій щодо застосування методів і засобів захисту веб-додатків від SQL-ін'єкцій.

Методи дослідження – опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння.

Захист від SQL-ін'єкцій має важливе значення в галузі кібербезпеки. SQL-ін'єкція-це тип атаки на бази даних, що дозволяє зловмисникам виконувати небезпечні запити до сервера та отримати несанкціонований доступ до інформації. Такі атаки залишають загрозою для веб-додатків особливо за недостатньої перевірки введених даних.

У роботі досліджено проблему захисту інформаційного ресурсу від SQL-ін'єкцій визначено мету та тему дослідження. Проведено аналіз основних типів SQL-ін'єкцій та розглянуто підходи захисту від SQL-ін'єкцій. Також проаналізовано основні методи запобігання та виявлення таких атак.

У роботі розглянуто інструмент ModSecurity який забезпечує захист веб-додатків.

Проведено практичне використання системи захисту ModSecurity, яке показало ефективність даного підходу для забезпечення захисту інформації. Визначено основні правила для виявлення SQL-ін'єкцій в реальному часі, та їх блокуванння.

На основі даних досліджень сформульовані рекомендації для покращення захисту веб-додатків від SQL-ін'єкцій, а також практичні заходи для існуючих IT-інфраструктур

Галузь використання – кібербезпека інформаційних ресурсів організації.

ІНФОРМАЦІЙНІ РЕСУРСИ, ВЕБ-ДОДАТКИ, SQL-ІН'ЄКЦІЇ, ЗАХИСТ ВІД АТАК MODSECURITY,МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ

ЗМІСТ **Стор**

7

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

WAF-Web Application Firewall

IDS- Intrusion Detection System

IPS-Intrusion Prevention System

HTTP -HyperText Transfer Protocol

IP- Internet Protocol address

SQL- Structured Query Language Injection

OWASP - Open Web Application Security Project

ORM- Object Relational Mapping

URl - Uniform Resource Locator

ВСТУП

Актуальність дослідження. Захист інформаційного ресурсу від SQL-ін’єкцій, є надзвичайно важливим у сучасних умовах розвитку веб-технологій. SQL-ін’єкція-один з найпоширеніших типів атак, який використовує базу даних для вразливості веб-додатків. SQL-ін’єкція дозволяє зловмисникам виконувати SQL-запити, змінювати і видаляти дані, отримувати конфіденційну інформацію, іноді навіть отримувати контроль над сервером.

Вразливості даного типу атаки, виникають загалом через недоліки в обробці в введені даних користувачами. SQL-ін’єкції-найбільш небезпечні для організацій, які зберігають фінансові дані користувачів, та персональні дані в базах даних.

У звя’зку з цим, дослідження має велике значення для впровадження ефективних методів і засобів захисту від SQL-ін’єкції. Один з розповсюджених інструметом в цій галузі є ModSecurity- це система (WAF), яка захищає чайт від спроб зламати його. Він сканує HTTP-запити до сервера та блокує IP-адреси з яких приходять підозрілі запити.

Вищесказане, доводить актуальність теми кваліфікаційної роботи, зміст якої полягає у дослідженні методів та засобів захисту інформаційного ресурсу від SQl-інєкцій.

Об’єкт дослідження – процес забезпечення захисту інформаційного ресурсу від SQL–ін'єкцій.

Предмет дослідження – методи та засоби захисту інформаційного ресурсу від SQL–ін'єкцій

Мета роботи є застосунок системи захисту інформаційного ресурсу від SQL-ін’єкцій, реалізований за допомогою інструментів безпеки, таких як ModSecurity, та формування рекомендацій щодо застосування методів і засобів захисту веб-додатків від SQL-ін’єкцій.

Наукові завдання:

дослідити принцип роботи SQL-ін`єкції та їх виконання;

проаналізувати атаки та уразливості SQL-ін`єкцій;

проаналізувати існуючих рішень для захисту інформаційного ресурсу від SQL-ін’єкції;

дослідити класичні методи SQL-ін`єкції;

встановити захист від SQL-ін’єкції за допомогою ModSecurity;

проаналізувати виявлення спроб атак SQL-ін’єкції за допомогою ModSecurity;

розробити рекомендації щодо підвищення захисту з використанням ModSecurity;

Методи дослідження- опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння.

Практичне значення одержаних результатів- запропоновані практичні рекомендації щодо захисту інформаційних ресурсів, від SQL-ін’єкцій із застосуванням ModSecurity, що можуть використовуватися фахівцями з кібербезпеки у процесі безпечної архітектури веб-ресурсів.

Результати кваліфікаційної роботи апробовані на Всеукраїнській науково-практичній конференції «Цифрова трансформація кібербезпеки», яка відбулася 24 квітня2025 року в Державному університеті інформаційно-комунікаційних технологій, м. Київ.

1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ЗАХИСТУ ІНФОРМАЦІЙНОГО РЕСУРСУ ВІД SQL-ІН’ЄКЦІЇ

1.1 Аналіз принципу роботи SQL-ін’єкції

SQL-ін'єкція (Structured Query Language Injection) – це один з найбільш часто використовуваних способів взлому продуктів, що взаємодіють з базами даних. Суть таких ін'єкцій – впровадити в дані (передані через GET, POST запити) будь-який випадковий SQL код. Якщо ресурс приймає і виконує такі ін'єкції, значить він вразливий, і, по суті, з базою даних можна поводитися, як завгодно (рис 1.1).

Особливу небезпеку SQL-ін'єкції становлять для інформаційниї ресурсів, які обробляють конфеденційні дані користувачів наприклад : особисті дані, фінансова інформація, облікові записи ітд. Основною причиною вразливості найчастіше є відсутність належної перевірки введення даних, або використання небезпечних методів формування SQL-запитів у коді веб-додатка.

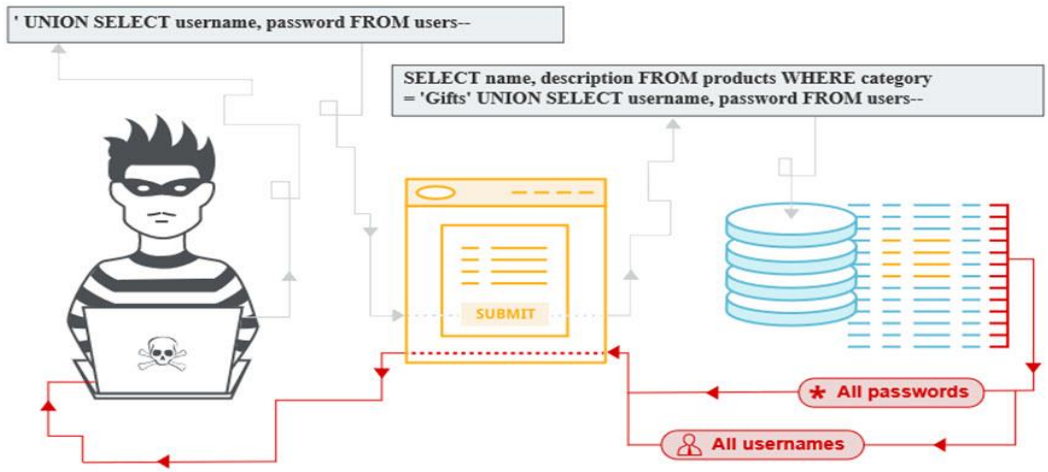

Рис 1.1 Приклад SQL-інєкції

За звітом OWASP (Open Web Application Security Project), SQL-інєкції входять в десятку найнебезпечніших вразливостей безпеки веб додатків. Незважаючи на велику кількість рішень для запобігання даних атак, більшість систем залишаються вразливими через людські помилки, відсутність оновлень ПЗ, або недосконалий контроль доступу.

Сучасні методи захисту від SQL-інєкцій:

Підготовлені запити та ORM-фреймворки:

Підготовлені запити – це конструкціїї SQL, у яких SQL-команда компілюється, а дані користувача передаються окремо як параметри. Завдяки цьому є можливість впровадити шкідливий код в структуру SQL-запиту. Використання ORM-фреймворків (наприклад, SQLAlchemy) також сприяє захисту, так як відокремлює роботу з базою даних на рівні об’єктів, зменшуючи потребу ручного написання SQL-коду.

Використання веб-фаєрволів (WAF),таких як ModSecurity

Web Application Firewall (WAF)- це компонент безпеки, який контролює мережний трафік, що надходить і виходить із бекенд-системи. Міжмережний екран застосунків (application firewall) спеціально контролює вхідний та вихідний трафік, доступ до програмних застосунків чи сервісів, працює з трафіком (HTTP/S, TCP, UDP тощо), який надходить чи виходить із вебзастосунків. Наприклад, ModSecurity- доволі популярний WAF з відкритим кодом, який інтегрується Apache, Nginx та іншимим веб-серверами.

Аудит коду та впровадження принципів безпечної розробки

Регулярний аудит коду дозволяє виявити уразливості на ранніх стадіях. Крім того важливо дотримуватися Secure Coding Practices, зокрема:

Мінімалізація використання динамічного SQL;

Ведення журналу підозрілих спроб;

Навчання розробників основам безпеки;

Обмеження прав доступу для користувачів додатку;

Фільтрація та валідація вхідних даних

Перевірка та очищення введених даних користувачами, є важливим методом захисту

Перевірка типів даних

Видалення або екранування спец символів (“,’,--,;) які можуть змінити структуру запиту

Використання білого списку дозволених шаблонів або символів

Регулярне тестування безпеки за допомогою допоміжних інструментів (SQLMap)

SQLMap — це безкоштовний інструмент з відкритим кодом для тестування на проникнення, який автоматизує процес виявлення та експлуатації вразливостей SQL-ін’єкцій, а також дозволяє отримати контроль над серверами баз даних. Цей інструмент має потужний механізм виявлення, багато спеціалізованих функцій для фахівців з тестування на проникнення та широкий спектр можливостей: від ідентифікації бази даних до вилучення даних, доступу до файлової системи і навіть виконання команд операційної системи через позаканальні з’єднання. Найбільш поширені методи SQl-ін’єкції також наведені в таблиці 1.1

Таблиця 1.1

Найбільш поширені методи

№	Методи захисту	Опис	Приклад реалізації
1	Підготовлені запити	Автоматично відокремлюють SQL-код від даних користувача запобігаючи ін’єкції.	Java або cursor.execute() з параметрами у Python.
2	ORM-фреймворки	Зменшують ризик ручного написання SQL-коду.	Django ORM, SQLAlchemy, Hibernate.
3	Веб-фаєрволи WAF	Блокують підозрілі запити та аналізують наявність шкідливих шаблонів.	ModSecurity.

4	Аудит коду	Перегляд коду з точки зору безпеки, дотримання стандартів безпечного програмування .	Code review.
5	Фільтрація та валідація даних	Перевірка типу, формату даних	Регулярні вирази, бібліотеки типу Joi

1.2 Аналіз типових атак та уразливостей інформаційного ресурсу від SQL-ін’єкції

SQl-ін’єкція є одним з найнебезпечніших типів атак на веб-додатки, яка дозволяє зловмисникам взаємодіяти з базою даних цих додатків, шляхом вставлення шкідливих SQL команд у поля введення даних або URl-запитию. Ці атаки можуть привести до витоку персональних даних, або їх модифікації, а також отримання доступу до системи.

За даними Acunetix основна причина їх виникнення – неналежна обробка вхідних даних. Якщо введеня користувача безпосереднь вбудовується в SQL-запити без фільтрації або параметризації, база даних сприймає ці дані як частина коду.

Типові напрямки атаки:

Поля входу (логін пароль);

Пошукові запити ;

Cookie-файли або HHTP-заголовки;

URL-параметри;

API-запити(XML)

Приклад шкідливого запиту:

SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '';

Цей запит завжди повертатиме істину, надаючи зловмиснику несанцкіонований доступ до аккаунту

Таблиця 1.2

Основні типи SQL-ін’єкцій

№	Тип SQL-ін’єкції	Опис	Приклад запиту
1	Класична	Вставка SQL-коду у запит	` OR `1`=`1
2	Сліпа (Blind SQLi)	Сервер не повертає помилок, але поведінка змінюється. Викори-стовуються логічні оператори	` AND 1=1 --
3	Out-of-Band SQli	Дані передаються через сторонні канали(DNS-запити)	`; Exec Xp_cmdshell(`nslookup
4	Uninon-Based SQLi	Об’єднання декількох SELECT-запитів для отримання дод даних	` Union select username, password
5	Time-Based Blind SQLi	Атака основується на затримках(Sleep())зоб визначити істиність тверджень	` or if (1=1, sleep(5), 0)--

Схема SQL-ін’єкції

Зображення 1.2 демонструє, як зловмисник надсилає шкідливий запит до веб-серверу через URL (teacherId=117 or 1=1), що приводить до виконання зміненого SQL-запиту (SELECT * FROM teachers WHERE teacherId=117 or 1=1;) В результаті

сервер повертає всі записи з таблиці, і вся інформація потрапляє до зловмисника в результаті чого може бути використана.

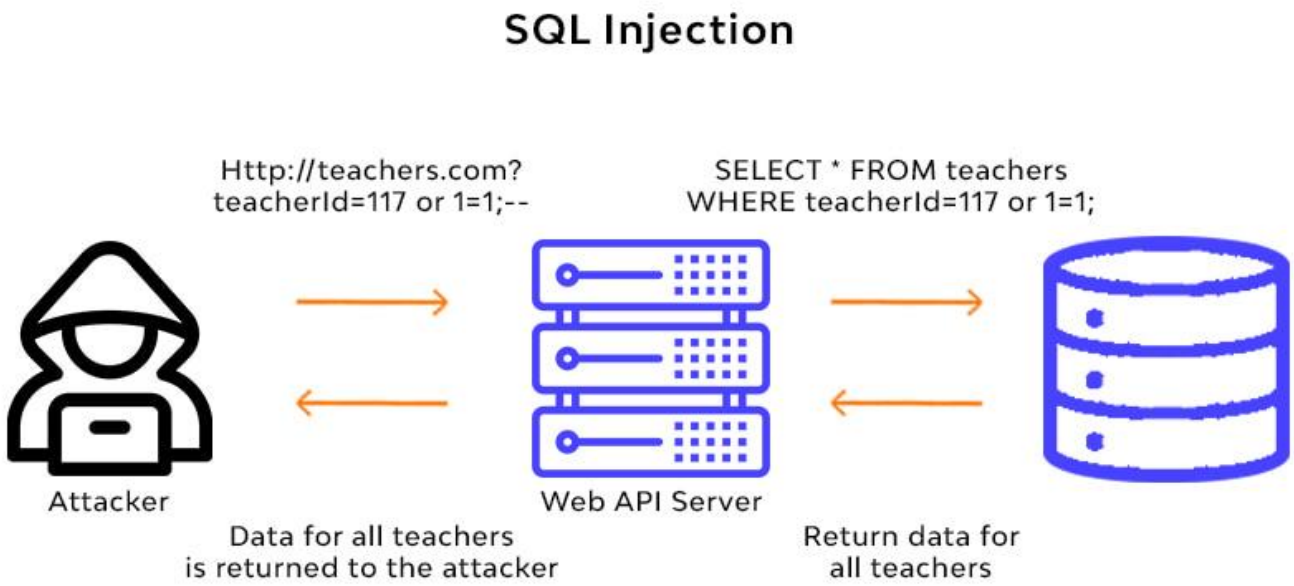

Рис 1.2 схема SQL-ін’єкції

Уразливі місця в системах:

1.Відсутність фільтраціх вхідних даних

Дані що вводяться користувачем (логін, пароль, прпсетри URL), не проходять перевірку на допустимість, зловмисник може впровадити SQL-код, замість звичайного тексту. Це змінює логіку SQL-запиту та дозволяж отримати доступ до конфіденційних даних користувача.

Наприклад:

Користувач вводить в логін ` OR 1=1 –

Запит стає SELECT * FROM users WHERE username = `` OR 1=1--` AND password = `…..` В результаті цього система надає доступ безпосередньо без введення та перевірки пароля.

2. Пряме вбудування змінних у SQL-запити

Найбільш небезпечне формування SQL-запитів шляхом їх додавання в рядок запиту що створює досконалі умови для SQL-ін’єкцій.

Наприклад:

Sql = “SELECT * FROM users WHERE id = “ + userId;

В такому випадку якщо userId = 1 OR 1=1 тоді запит вибирає всіх користувачів.

3.Відсутність використання Prepared Statements

Prepared Statements або підготовлені запити дозволяють бачити чітку межу між SQL-кодом та вхідними даними, що робить неможливим виконання SQL-коду, введеного корустувачем.

Наприклад:

PreparedStatement stmt = conn.prepareStatement(“SELECT * FROM users WHERE id =?”);

Stmt.setInt(1, userId);

4.Публічний доступ до інтерфейсів адміністрування

Адмін-панелі з відкритим або слабо захищеним доступом (без авторизації або стандарт логінами) є вразливі для SQL-ін’єкцій або інших атак.

Наприклад:

Якщо /admin/ не має CAPTCHA обмежень, зловмисник може використати SQL-ін’єкцію для входу в адміністратора. Рішенням цього може бути (обмеження доступу за IP, двофакторна автентифікація, приховання адмін інтерфейсів)

5. Використання облікових записів з надмірними привілеями

Дуже часто застосунки підключають бази даних з обліковим записом який має повні root права. Якщо SQL-ін’єкція буде успішною зловмисник отримає повний контроль над базою даних: видалення, зміна, вставка даних.

Наприклад:

Якщо обліковий запис має права DROP,UPDATE,DELETE,INSERT, замість SELECT тоді потрібно виділити окремі облікові записи для читання, запису, або

мінімалізувати доступ принципу найменших привілеїв

Наслідки SQL-ін’єкції:

Отримання адміністративного доступу до системи

Через SQL-ін’єкцію хакер може обманути базу даних і ввійти в систему як адміністратор, навіть якщо зловмисник не буде знати паролю. Даним способом він зможе змінювати налаштування, додавати нових користувачів, вимикати захист і видаляти таблиці даних

Витік або модифікація критично важливої інформації

Зловмисник може викрасти все що зберігається в базі даних : паролі, дані користувачів, банківські карти, повідомлення і тд.

Знищення даних в базах

Зловмисник може стерти важливу інформацію. У разі цього компанія втратить дані які могла накопичувати роками .

Порушення роботи сервісу, репутаційні та фінансові втрати

SQL-ін’єкція виводить з ладу роботу сайту, мобільного додатку або якусь внутрішню систему і користувачі не зможуть зайти і скористатися послугами або замовити товар.

Тому навіть найменша ‘дірка’ в захисті яка пов’язана з SQL-ін’єкцією, може нести за собою наслідки. Тому в сьогоденні важливою частиною є безпечна обробка запитів до бази даних

1.3 Аналіз існуючих рішень для захисту інформаційного ресурсу від SQL-ін’єкції

На сьогодні існує багато технологічних рішень, які дозволяють ефективно запобігати атаки по типу SQL-ін’єкцій. Ці засоби захисту можна поділити на декілька категорій: програмні методи, веб-аплікаційні брандмауери (WAF), інструменти для автоматизованого тестування безпеки, а також методи безпечного програмування.

Використання підготовлених запитів (Prepared Statements)

Один із найефективніших способів захисту які існують на даний момент від SQL-ін’єкції- це використання підготовлених (параметризованих) запитів. Цей метод дозволяє чітко розділити SQL-логіку та дані користувача, що робить майже немоливим виконання шкідливого SQL-коду у базі даних. Майже у всіх сучасних мовах програмування (PHP, Java, Python) є вбудована підтримка механізмів використання підготовлених запитів через ORM, або драйвери БД.

Перевагою таких запитів є: Надійний захист від SQL-ін’єкції та простота їх реалізації

Недоліком використання підготовлених запитів є те що: Вони потребують правильної реалізації на всіх рівнях коду.

ORM (Object-Relational Mapping) фреймворки

ORM-фреймворки наприклад: Hibernate (Java), Entity Framework(.NET), Django ORM(Python), автоматично відсліжують вхідні дані та формують безпечні SQL-запити. Хоча 100% захист гарантувати вони не можуть, але їх використання значно зменшує ризики, пов’язані з ручним формуванням SQL-запитів.

Перевагою фреймворків є: простота коду можна працювати з базою як з об’єктами, ORM автоматично використовує підготовлені запити або параметри, підвищує продуктивність розробки вручну не потрібно писати SQL,також ORM мають вбудовані інструменти для управління схемою бази даних.

Недоліком фреймворків є: Менший контроль над SQL-запитами ORM може генерувати неефективні запити у великих проектах ORM може бути повільним якщо порівнювати з ручними SQL-запитами. Також неправильне їх використання може призвести до вразливостей.

Веб-аплікаційні фаєрволи WAF

WAF-Web Application Firewall- це спеціальний засіб безпеки, який встановлюється між користувачем і веб-додатком. Його завдання – аналізувати трафік, що надходить до вебсайту, і виявляти потенційно небезпечні запити, перш ніж вони досягнуть серверної частини. Коли користувач надсилає запит (відкриває сторінку чи авторизується) він перевіряє вміст цього запиту- шукає в ньому ознаки SQL-ін’єкції та інших атак. Якщо він находить підозрілі запити то записує це все в інцидент журнал, попереджує адміністратора, блокує запит.

Перевагою WAF є : Миттєве виявлення і блокування атак, не вимагає змін у коді, дозволяє створювати правила під той сайт що тобі потрібно, фіксує всі підозрілі дії що допомагає в рослідуванні.

Недоліком є: Налаштування цим методом потребує досвіду, неправильне налаштування може блокувати простих користувачів. Впливає на продуктивність тому що обробка всіх запитів займає багато ресурсів.

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ЗАХИСТУ ІНФОРМАЦІЙНОГО РЕСУРСУ ВІД SQL ІН'ЄКЦІЇ

2.1 Класичні методи захисту від SQL-ін'єкції

SQL-ін'єкція залишається одним з небезпечних тип атак на веб-додатки, оскільки дозволяє отримати несанкціонований доступ до баз-даних, та отримати або знищити конфіденційні дані. Для запобігання цього застосовують класичні методи захисту, які повинні бути обов'язковими на всіх рівнях розробки та впровадження в веб-системи.

1.*Параметризовані(підготовлені) запити (Prepared Statements)*

Це один з найбільш надійних способів захисту від SQL-ін'єкції. Суть цього методу полягає в тому що SQL-запит компілюється сервером бази даних, після чого в нього передаються лише параметри- дані введені користувачем, значення, які не можуть змінити структуру запиту.

Приклад PHP з PDO:

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE email = ?");

$stmt->execute([$email]);
```

В наведеному випадку значення $email не виконується безпосередньо SQL-запитом, а передається як параметр, що не дає можливості щоб виконався шкідливий код.

Приклад Java

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE id = ?");
```

ps.setInt(1, userId);

Python

cursor.execute("SELECT * FROM users WHERE username = ?", (username,))

У всіх наведених випадках структура чітко відділена від запиту який вводить користувач що забезпечує доволі надійний захист.

Фільтрація та валідація даних

Фільтрація даних – це очищення даних які вводить користувач від небезпечних символів, які використовуються для SQL-ін’єкції. Валідацією є перевірка змісту введених даних та, формату щоб переконатися що вони відповідають очікуваних даних(наприклад число дата,email).

Основними правилами є:

Перевірка типу: номер телефону, дата;

Очищення системи від спец символів.

Числові значення перетворювати в тип Int.

Встановлювати максимальну довжину рядків .

Приклади:

У PHP використовують Filter_Var().

У JavaScript- RegExp.

У Python використовують Cerberus,Pydantic.

Застосування білого списку дозволених значень замість чорного списку заборонених – вважається кращою практикою

Валідація на стороні сервера

Багато розробників покладаються на клієнську валідацію через JavaScript. Це є помилкою, бо атакувальник може змінити або обійти JS-перевіркою. Із-за цього всі перевірки повинні виконуватися зі сторони сервера.

Сервер має перевіряти тип і допустимі значення, забезпечити відповідність формату, виконувати логічні перевірки.

Це дозволяє зупинити атаки ще до сформованого SQL-запиту.

Обмеження прав доступу до баз даних

Обліковий запис, яким веб-додаток підключається до бази даних, не повинен мати прав адміністратора. Він повинен мати лише дозволи, які потрібні для його нормального функціонування. Якщо додаток лише читає дані – обліковий запис повинен мати лише Select без Insert, Update, Delete.

Логування підозрілих запитів

Запис усіх підозрілих SQL-запитів та помилок до лог-файлів дозволяє швидко виявити підозрілу активність. У подальшому це також дозволяє аналізувати інциденти.

Типові ознаки

Наявність у запитах підозрілих символів;

Повтори одинакових запитів;

Звернення до службових таємниць

На основах логів можна створювати автоматичні правила для блокування сповіщення

Аудит коду

Регулярний аудит коду дозволяє виявити місця, де найчастіше формуються SQL-запити з введених даних, і перевірити їх безпечну реалізацію. У великих компаніях аудит коду часто автоматизують за допомогою SAST(Static Application Security Testing)- Інструмент який аналізує код без його запуску.

Використання шаблонів безпечного програмування

Розробка повинна базуватися на принципах Secure coding practices.

Уникнення динамічного SQL;

Ізоляцією логіки обробки записів від бази;

Використання шаблонів MVC;

Суворе розмежування доступу;

Використання ORM-фреймворків;

ORM - технологія яка дозволяє працювати з базою даних, за допомогою об’єктів а не запитів- SQL. Це облегшує спеціалістам ручного написання SQL-коду, як наслідок, знижує ризик SQL-ін’єкції.

Популярні ORM-фреймворки:

Django ORM (Python)

SQLAlchemy (Python)

Hibernate (Java)

Eloquent ORM (PHP)

ORM екранує автоматично вхідні дані та використовує підготовлені запити.

Наприклад Django: User.objects.filter(username=username)

Тут обробляється безпечно username, навіть якщо користувач вводить потенційно небезпечні символи. Але його неправильне використання знову відкриє можливість для атак.

Реалізація політик безпеки на серверному рівні

Крім захисту коду додатку, важливо також реалізувати захисні механізми на рівні сервера баз даних. Наприклад, багато сучасних СУБД, дозволяють створювати політики, які реагують на підозрілі запити.

Приклад таких політик:

Заборона таких політик Drop table, Alter, Grant у продуктивному середовищі.

Обмежити виконання DLM-запитів поза робочим часом.

Сповіщувати адміністратора при надмірних запитах Select-запитів від одного користувача.

Також потрібно налаштовувати журнали аудиту, які фіксують спроби до доступу критичних таблиць та формують сигнали безпеки для Siem-систем.

Рекомендації OWASP класичного захисту

Проект OWASP створює та оновлює перелік найнебезпечніших вразливостей. SQL-ін’єкція традиційно входить до цього списку. Серед рекомендацій:

Використовувати safe APIs які повністю деактивують прямі SQL;

Уникати повідомлення про помилки які містять SQL-код;

Проводьте регулярне пентестування та статичний аналіз коду.

Використовуйте whitelist- валідацію для введення.

OWASP також рекомендує безкоштовні інструменти для розробників зокрема:

ZAP, ESAPI.

Таким чином класичні методи від SQL-ін’єкції охоплюють як технічні інструменти розробника, так і загальні принципи веб-додатків. Їх поєднання забезпечує надійний бар’єр між системою та зловмисниками.

2.2 Використання спеціалізованих засобів захисту (WAF, IDS/IPS)

Використання спеціалізованих засобів захисту (WAF, IDS/IPS) є обов’язковим мінімумом для створення безпечного веб додатку, так як класичних методів може бути недостатньо. Отже для побудови безпечної кіберзахищеної стійкої системи потрібно використовувати спеціалізовані засоби: WAF(Web Application Firewall), IDS (Intrusion Detection System), IPS(Intrusion Prevention System).

1WAF(Web Application Firewall)

WAF- це тип брандмауера, який налаштований для роботи спеціально з веб-додатками. Основна його задача ‘спостерігати’ за двостороннім веб трафіком (HTTP/HTTPS), що переміщується між веб додатками та інтернетом, виявляючи та блокуючи шкідливих користувачі, перш ніж вони попадуть до веб-додатку.Waf робить це шляхом фільтрації, моніторингу та блокування шкідливого трафіку та атак на рівні додатків.

Основні методи WAF які використовують для фільтрації та видалення шкідливого трафіку на рівні додатків:

Blocklist WAFs: Підхід блокує певні тип шкідливого трафіку, а не точні джерела.

Allowlist WAFs: Цей метод за замовчуванням зупиняє весь трафік дозволяючи проходити лише підтвердженому трафіку. Хоч і це найбільш безпечний підхід, але також він може затримати непередбачений легітивний трафік.

Hybrid WAFs: Даний метод об’єднує модель одночасного і блокування та дозволу одночасно

WAFs є корисними проти атак, таких як підборка міжсайтових запитів, включеня файлів, SQL-ін’єкції.

Крім того, WAF може забезпечити симетричну фільтрацію шляхом очищення не лише вхідних запитів, але й вихідного трафіку. Зловмисний вихідний трафік може генеруватися, наприклад, якщо в мережі інфіковано один з комп’ютерів. У цьому разі комп’ютер може почати комунікувати з командним сервером ботнету або брати участь у DDoS-атаці. WAF здатний заблокувати цю діяльність і повідомити адміністраторів про проблему.

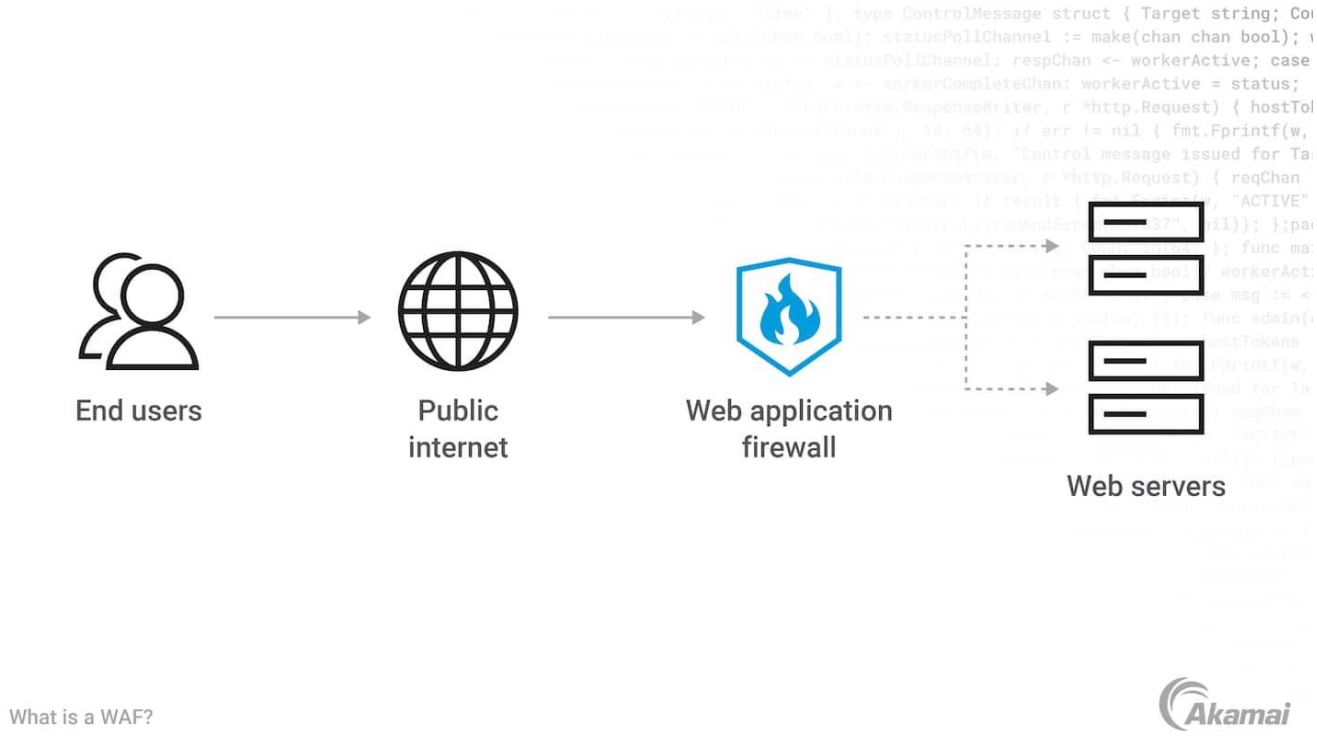

Рис 2.1 Схема роботи WAF

ModSecurity

ModSecurity-це WAF з відкритим кодом який є міжмережевим екраном з відкритим вихідним кодом що забезпечує захист від атак спрямованих на веб-програми. ModSecurity дозволяє здійснювати моніторинг трафіку та виконує аналіз подій в

реальному часі. На сьогодні ModSecurity успішно виконують захист Web-серверів у всьому світі.

Приклад для блокування SQL-ін’єкції

SecRule ARGS "@rx (\b(select|union|insert|update|delete|drop)\b)" \

"phase:2, id:1001, deny, msg:'SQL Injection Attempt Detected'"

Дане правило сканує всі параметри запиту (ARGS) на наявність підозрілих ключових SQL-слів і блокує запит.

Комерційні WAF

Комерційні WAF(Web Application Firewall)- це спеціалізовані системи захисту вебзастосунків які продаються та підтримуються комерційними компаніями.

Переваги комерційних WAF:

Автоматичне оновлення сигнатур;

Аналітика та звітність;

Підтримка AI для виявлення аномалій;

Інтеграція з SIEM;

Недоліком є:

Вартість;

Складність налаштування;

Можливість False positives,що впливають на користувачів;

IDS/IPS-системи

IDS та IPS - це системи для належного моніторингу можливих загроз. Ці системи захисту від вторгнень абсолютно різні.

IDS- або система виявлення вторгнень зазвичай пов’язана з програмним забезпеченням або пристроєм який відповідає за моніторинг онлайн загроз. Незалежно від того чи находиться він в системі або мережі. Інформація про різні типи атак зазвичай надсилається адміністратору або серез систему SIEM.

Типи IDS:

HIDS- Система належного моніторингу ОС, на пристроях та хостах. Усі пакети відстежуються виявлення незвичайної активності. Сповіщують при критичних змінах файлів в системі ;

NIDS- Прослуховує вхідний трафік. Ця система аналізує активність трафіку, який надходить на/з пристрою. Вся активність порівнюється з наявною бібліотекою можливих атак, і як тільки виявляється щось дивне, адміністратор отримує сповіщення про це.;

IDS може моніторити комп'ютер, а також мережеву активність і піля цього розділяти активність на аномальну і нормальну. Цей тип системи був розроблений для пошуку дивної активності. Підхід зазвичай для цього типу є машинне навчання завдяки якому запам'ятовуються правила моделі активності, і порівнюються з новими/дивними. Цей підхід дуже хороший завдяки узагальненим властивостям.

Виявлення IDS на основі підпису. Цей традиційний підхід використовується на пошуку певних шаблонів. Таке виявлення IDS допоможе в бородьбі з існуючими кібератаками, але матиме багато проблем з шаблонами.

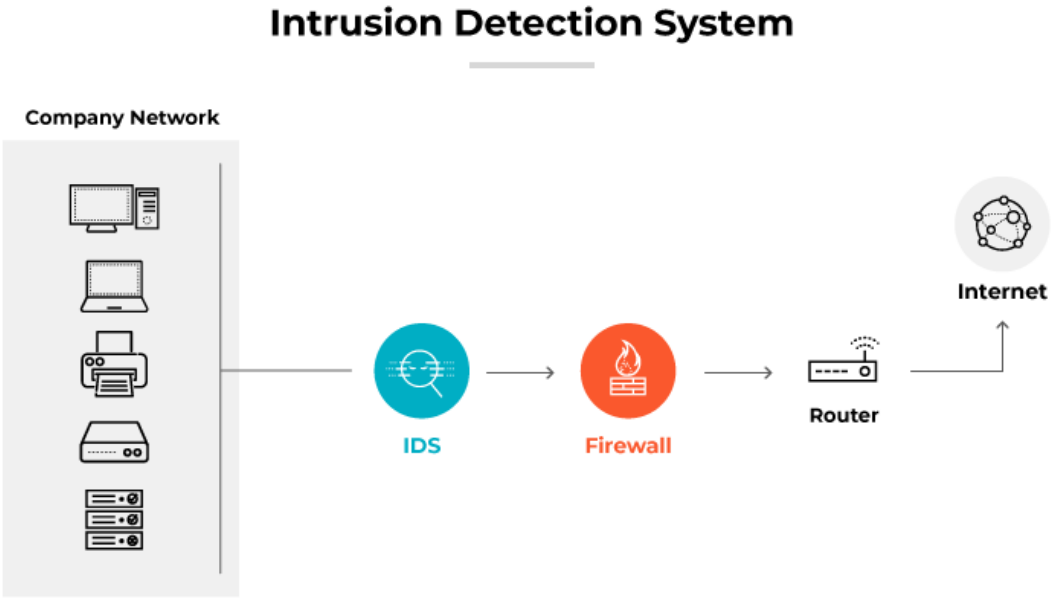

Рис 2.2 Схема роботи - Intrusion Detection System

IPS-система запобігання вторгненням функціонує шляхом виявлення загрозливої активності, повідомлення про таку активність і спроб запобігти таким загрозам. Як правило, IPS знаходиться відразу за брандмауером. Цей тип систем надзвичайно

корисний для виявлення проблем, пов'язаних зі стратегіями безпеки, виявлення кіберзлочинців та ідентифікації загроз документам.

Існує кілька типів IPS:

Моніторинг аналізу протоколів. Цей метод функціонує порівнянням всієї активності з узагальненими правилами.

Моніторинг на основі підпису. Цей метод виявляє пакети в мережі та порівнює їх з стандартними шаблонами.

Моніторинг на основі статистичних даних. Перевіряє мережеву активність та порівнює її на основі визначених заздалегідь базових ліній. Ця лінія визнаає основні характеристики, які вважаються нормальними.

Після виявлення підозрілої активності IPS може реагувати за сценаріями:

IPS може заблокувати певного користувача який порушує шаблони отримуючи доступ до мережі, хосту або програми.

Видалення загрозливого контенту або видалення заражених даних після атаки.

Класифікація типів IPS

WIPS. Цей тип IPS виявляє несанкціонований доступ і усуває його, як тільки помітив. Зазвичай така система функціонує як надбудова над інфраструктурою бездротової локальної мережі, але може використовуватися і самостійно. Завдяки використанню WIPS можна запобігти таким ризикам, як honeypot, несанкціоновані точки доступу, атаки на відмову в обслуговуванні, підміна MAC-адрес та багато інших.

HIPS. Ця система зазвичай працює шляхом аналізу поведінки коду на хості, і таким чином виявляється дивна активність. HIPS надзвичайно корисний для захисту конфіденційної інформації від вилучення.

NIPS. Виявлення відбувається шляхом аналізу пакетів через мережу. Одразу після встановлення збираються дані про хост і мережу. Запобігання атакам здійснюється шляхом відхилення пакетів, обмеження пропускної здатності та TCP-з'єднань.

NBA. Цей аналіз проводиться на основі нормальної/дивної поведінки в мережі. Для того, щоб розглянути, що є нормою, а що ні, системі потрібен певний час.

HIDS. Така система необхідна для належного моніторингу ОС на пристроях або окремих хостах. Всі пакети відстежуються, щоб виявити незвичну активність. Сповіщення відбувається, коли в системі змінюються критичні файли

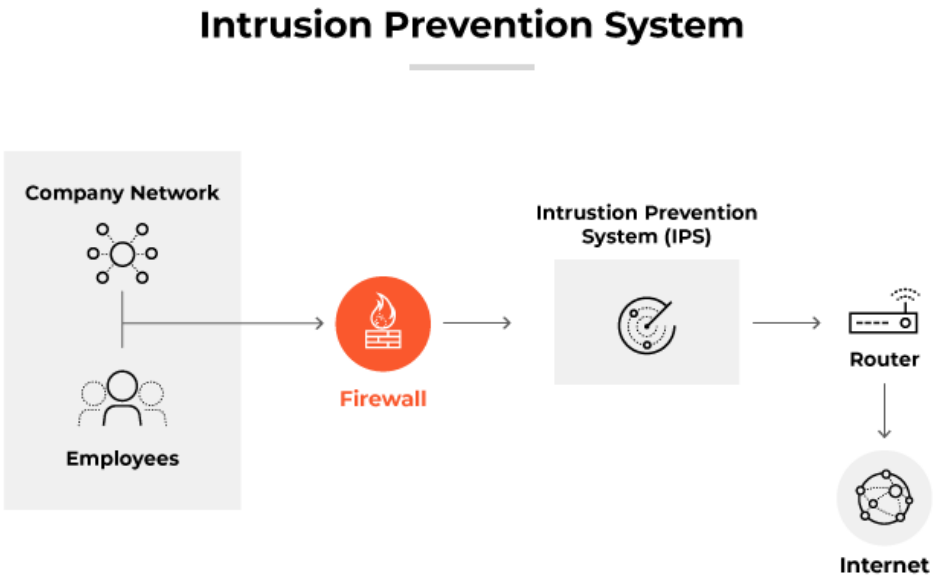

Рис 2.3 Схема роботи Intrusion Prevention System

Комбіноване використання IDS та IPS

IDS та IPS функціонують для досягнення спільної мети - захисту інфраструктури мережі. У більшості випадків ці системи виявляють дивну активність, порівнюючи її зі стандартними (нормальними) характеристиками поведінки.

Окрім незалежної роботи, IPS та IDS можуть бути інтегровані між собою. Більше того, брандмауери також можуть співпрацювати з IPS/IDS для кращого захисту системи. Така співпраця називається UTM або NGFW.

2.3 Порівняння ефективності існуючих рішень

В сучасному просторі інформаційних технологій застосовується дуже багато методів для захисту інформаційного простору від SQl-ін’єкції. Попри їхню загальну мету їхня ефективність значно варіюється в залежності від умов, правильності впроваджень, маштабів та кваліфікації персоналу.

1 Аналіз ефективності в різних ситуаціях

Для невеликих веб-додатків

У невеликих проєктах, де кількість точок введення обмежена, найкраще себе зарекомендували Prepared Statements. Вони мають низький бар'єр для впровадження і забезпечують практично повну ізоляцію SQL-коду від введення користувача. Отже базовий рівень захисту досягається мінімальними ресурсами.

Для середніх систем

Якщо система включає багато форм, фільтрів та активну взаємодію з базою, одного рівня недостатньо. Тут ORM (наприклад, Django ORM або Laravel Eloquent) забезпечує централізацію доступу до БД, що знижує ризики. При цьому валідація введення може забезпечити додатковий захист від нетипових векторів атак. Отже: ORM + валідація = баланс між зусиллями і безпекою.

Для великих/критичних систем

Високонавантажені або критичні системи (банки, держсектор, медичні бази) потребують многошарової архітектури захисту:

На прикладному рівні — Prepared Statements та ORM.

На прикладному периметрі — WAF.

На мережевому — IDS/IPS.

Для виявлення аномалій — SIEM або централізоване логування.

Тут ефективність кожного елемента залежить не лише від технологій, а й від регулярного аналізу та оновлення правил. Отже: тільки поєднання кількох рішень забезпечує прийнятний рівень стійкості.

Результати тестування

На базі демо-додатку було проведено симуляції атак за допомогою SQLMap, WAFNinja та ручних ін'єкцій.

Таблиця 2.1

Результати тестування симуляції атак

Конфігурація	Заблокованих атак	Виявлення в логах	Пропущених атак
Без захисту	0%	-	100%
Prepared Statements	100%	-	0%
Prepared + Валідація	100%	-	0%
Prepared + WAF	97%	часткове	3%
Prepared + WAF + IDS	98%	повне	2%
Повна схема + SIEM	99.5%	повне	<1%

Загальні критерії за якими будуть порівнюватися засоби захисту інформаційного простору від SQL-ін'єкції.

Рівень захисту- здатність рішення запобігати SQl-ін'єкції.

Складність впровадження в систему- можливість адаптації до змін в додатку.

Вартість впровадження та технічної підтримки.

Вірогідність помилкових спрацювань.

Гнучкість- можливість адаптації(змін) під певну систему.

Стійкість до обхідних технік.

Таблиця 2.2

Порівняння ефективності

Рішення	Рівень захисту	Гнучкість	Впровадження	Вартість	False Positives	Стійкість до обходу
Prepared Statements	Високий	Висока	Середня	Низька	Низька	Висока
Валідація введення	Середній	Висока	Низька	Низька	Середня	Низька
ORM	Високий	Середня	Середня	Середня	Низька	Висока
WAF (ModSecurity тощо)	Середньо-високий	Висока	Середня	Середня	Висока	Середня
IDS/IPS	Середній	Середня	Висока	Висока	Середня	Середня
Комбінований підхід	Дуже високий	Висока	Висока	Висока	Низька	Висока

Приклади обхідних технік

WAF або IDS часто виявляють лише шаблонні атаки. Проте обфускація (UN/**/ION, %27+OR+1=1--) або обхід через encoding (%2f, %5c) дозволяє уникати блокування. Тільки комбінація різнорівневих засобів дозволяє зменшити ці ризики.

3 РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ЗАСТОСУВАННЯ МЕТОДІВ ТА ЗАСОБІВЗАХИСТУ ІНФОРМАЦІЙНОГО РЕСУРСУ ВІД SQL-ІН'ЄКЦІЇ

3.1 Встановлення захисту від SQL-ін'єкції за допомогою ModSecurity

Загальні відомості про ModSecurity

ModSecurity — це відкрита система виявлення та запобігання веб-атакам, яка найчастіше використовується як веб-аплікаційний фаєрвол (Web Application Firewall — WAF). Вона підтримує фільтрацію, моніторинг та блокування HTTP-запитів у режимі реального часу. Основна мета ModSecurity — захист веб-серверів від різноманітних атак, зокрема SQL-ін'єкцій, XSS (міжсайтових скриптів), LFI (локальних включень файлів) та багатьох інших загроз, що відомі за класифікацією OWASP.

ModSecurity може бути встановлений на веб-сервери Apache, Nginx або IIS. Найпоширеніше його використання в Linux-середовищі разом з Apache. Він дозволяє створювати власні правила або імпортувати стандартні набори правил, наприклад OWASP Core Rule Set (CRS).

Підготовка до встановлення

Перед тим як почати встановлення ModSecurity, необхідно переконатися, що:

Сервер працює на ОС, що підтримує ModSecurity (Linux, Windows),

Встановлений веб-сервер Apache або Nginx,

Є доступ до root або sudo.

Як приклад розглянемо встановлення на Ubuntu Server 22.04 із веб-сервером Apache.

Встановлення ModSecurity (Apache + Ubuntu)

Оновлення системи:

sudo apt update && sudo apt upgrade

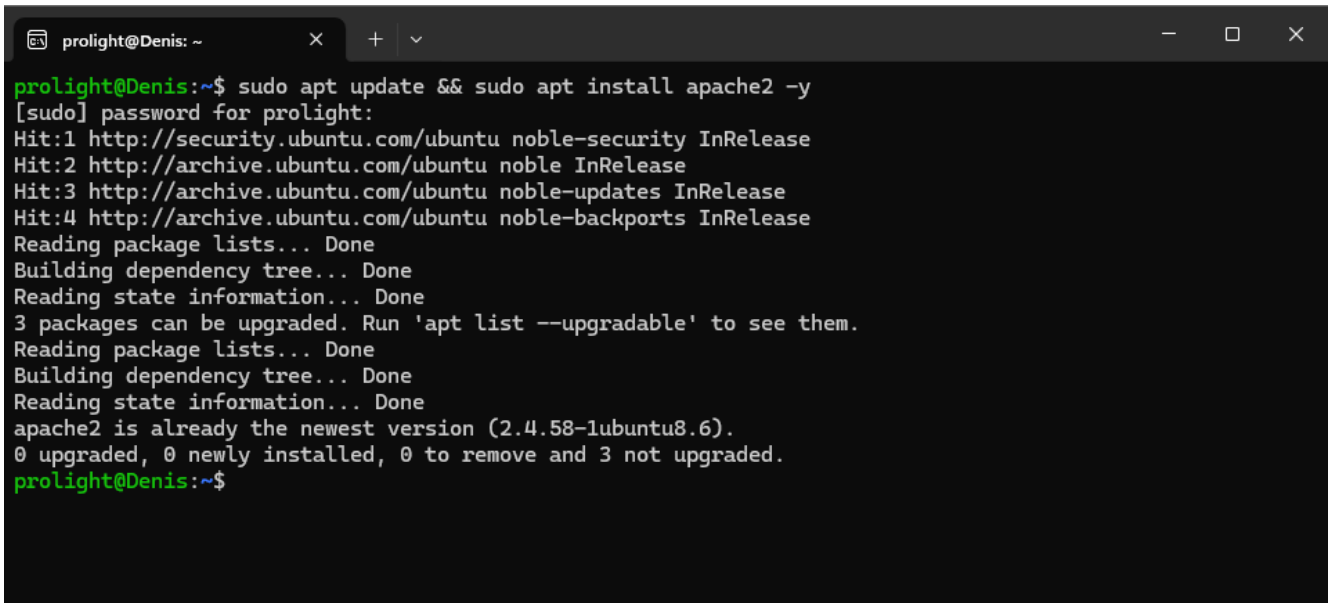

Рис 3.1 Оновлення системи

Встановлення ModSecurity

sudo apt install libapache2-mod-security2

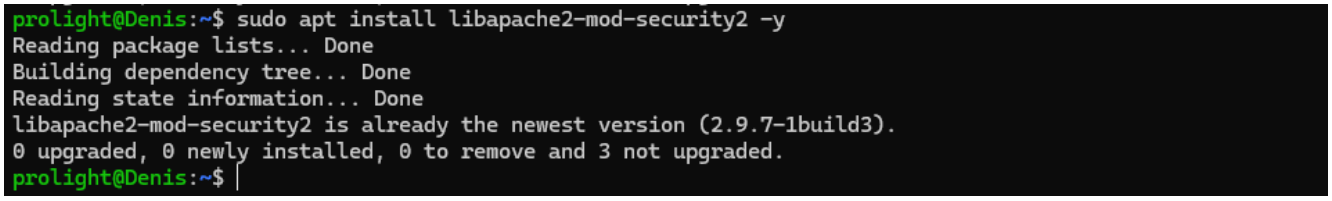

Рис 3.2 Встановлення ModSecurity

Активування модуля:

sudo a2enmod security2

Рис 3.3 Активування модуля:

Базова конфігурація ModSecurity

Основний конфігураційний файл:

/etc/modsecurity/modsecurity.conf

```
prolight@Denis:~$ sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.c
onf
```

Рис 3.4 Основний конфігураційний файл:

У ньому необхідно змінити директиву:

SecRuleEngine On

Це дозволить ModSecurity працювати у режимі блокування (тобто запити, що відповідають правилам, будуть блокуватися).

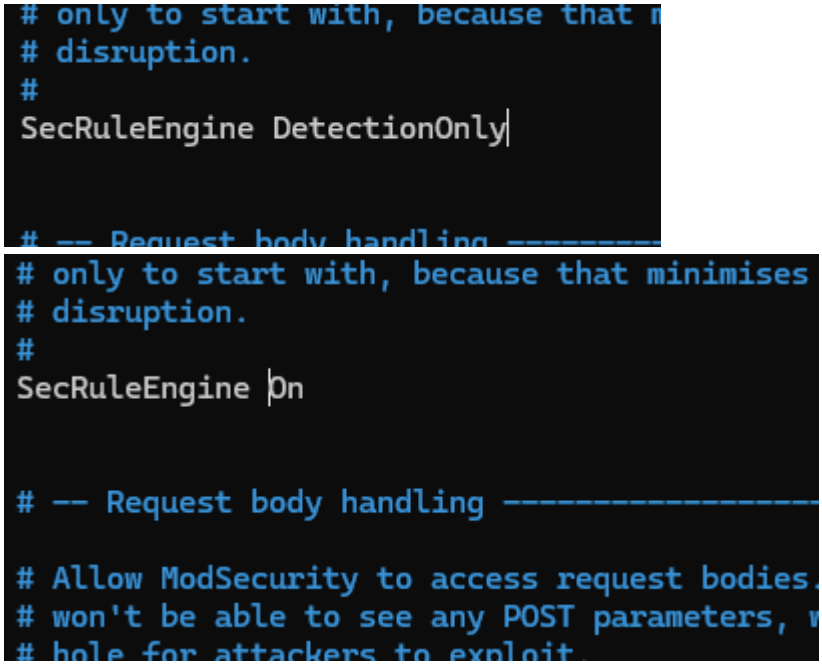

Рис 3.5 Зміна директиви

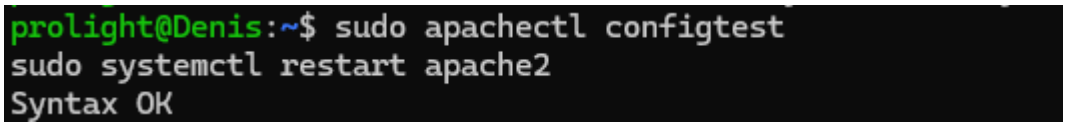

Рис 3.6 Перевірка результату

Реалізація правила захисту від SQL-ін’єкції вручну

Окрім стандартного набору правил, адміністратор може створювати власні специфічні правила для виявлення SQL-ін’єкцій.

SecRule ARGS "@rx \b(select|union|insert|update|delete|drop|exec|--)\b" \

"id:100001,phase:2,deny,log,status:403,msg:'Possible SQL Injection Attempt'"

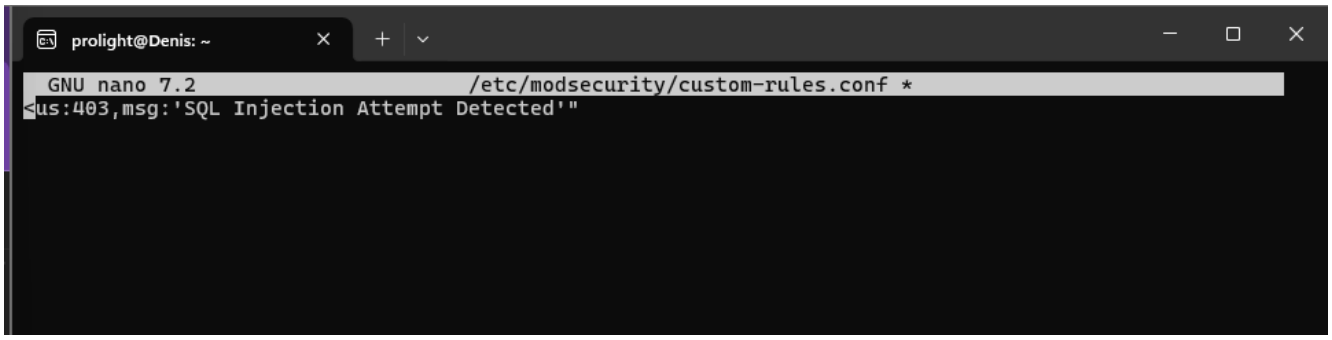

Рис 3.7 Реалізація правил

Тестування встановлення

Щоб переконатися, що ModSecurity працює, можна виконати простий тест:

Відкрити браузер і надіслати такий запит:

http://localhost/?id=1+union+select+1,2,3

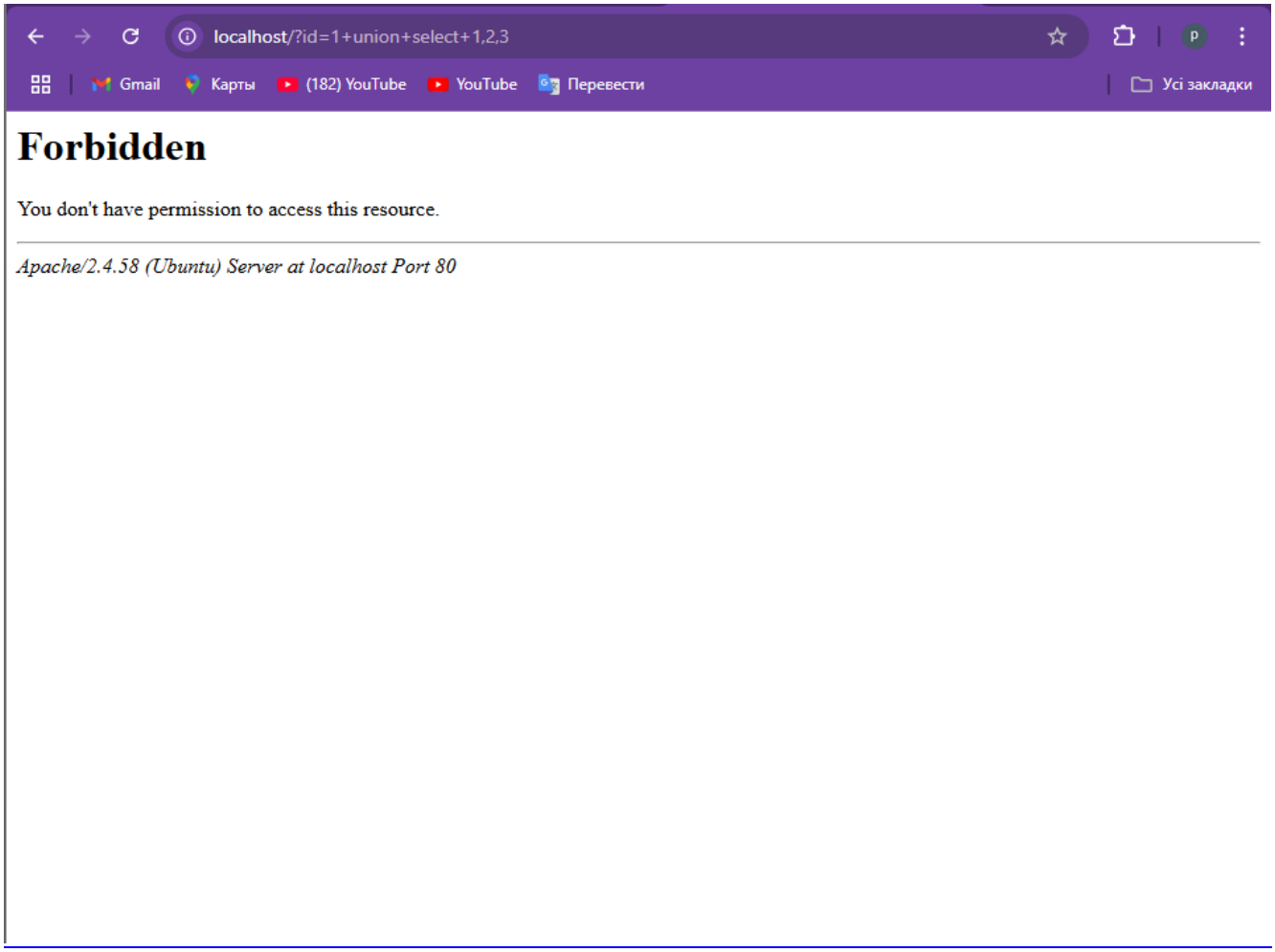

Рис 3.8 Тестування встановлення

Якщо правило працює, користувач побачить сторінку з помилкою 403 або власним повідомленням про блокування.

У логах /var/log/apache2/modsec_audit.log має з’явитися запис із повідомленням про SQL-ін’єкцію.

Ведення журналів подій

ModSecurity може зберігати всю активність у логах, що дозволяє:

Аналізувати спроби SQL-ін’єкцій,

Визначати джерела атак,

Формувати інцидентні звіти.

Логи зберігаються зазвичай у:

/var/log/apache2/modsec_audit.log

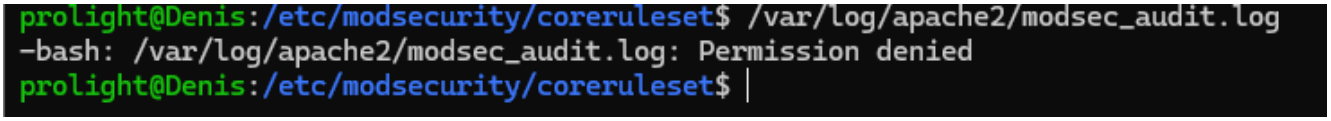

Рис 3.9 Перевірка логів

Типові проблеми при встановленні та їх вирішення

Таблиця 3.1

Типові проблеми при встановленні та їх вирішення

Проблема	Причина	Рішення
Apache не запускається після включення правил	Конфігураційна помилка	Перевірити синтаксис: apachectl configtest
Помилкове блокування користувачів	Надто агресивні правила	Перевірити лог, додати винятки
Відсутність журналу ModSecurity	Логування не ввімкнене	Перевірити SecAuditLog у конфігурації

Таблиця 3.2

Порівняння з іншими WAF-рішеннями

Характеристика	ModSecurity	Cloudflare WAF	AWS WAF
Тип	Open-source	SaaS	Хмарний
Гнучкість	Висока	Середня	Висока
Потребує налаштування	Так	Мінімальна	Мінімальна
Логи на сервері	Так	Ні	Опційно

ModSecurity особливо ефективний у поєднанні з іншими методами захисту, такими як prepared statements, IDS/IPS, SIEM та інші.

3.2 Процес виявлення спроб атак SQL-ін’єкції за допомогою ModSecurity

Одним із ключових завдань при захисті інформаційного ресурсу є виявлення спроб атак, спрямованих на базу даних — зокрема, SQL-ін’єкцій. Цей тип атаки дозволяє зловмиснику змінити поведінку SQL-запитів шляхом впровадження шкідливого коду в параметри HTTP-запитів. У даному підрозділі розглянуто практичний механізм виявлення SQL-ін’єкцій за допомогою модуля ModSecurity, а також оцінено ефективність створених правил та інтеграцію з правилами OWASP Core Rule Set.

Принцип виявлення SQL-ін’єкцій

Модуль ModSecurity діє як міжпроксі-фільтр для веб-сервера Apache та здатен аналізувати запити користувачів на ранніх етапах їх обробки. Виявлення SQL-ін’єкції може відбуватися шляхом:

сигнатурного аналізу (наприклад, пошук ключових слів "SELECT", "UNION", "DROP"),

детекції за допомогою вбудованої бібліотеки libinjection,

евристичного аналізу через систему правил.

Для цього у файлах конфігурації створюються правила, що обробляються під час фази аналізу HTTP-запиту. Якщо умова спрацьовує, модуль може заблокувати

запит (deny), записати інцидент у журнал (log) або лише повідомити (документувати).

Створення та налаштування власного правила

У межах дослідження було створено користувацьке правило для виявлення типових конструкцій SQL-ін’єкцій. Воно перевіряє наявність ключових слів у параметрах GET/POST:

Код правила:

SecRule ARGS "@rx (?i:(select|union|insert|delete|drop|--|;))" "id:9999999,phase:2,deny,log,status:403,msg:'Custom SQL Injection Rule Triggered'"

Коментарі до ключових параметрів:

SecRule ARGS — перевіряються всі аргументи запиту;

@rx — регулярний вираз для пошуку типових слів SQL;

id — унікальний номер правила (повинен не дублюватись);

phase:2 — правило спрацьовує під час аналізу тіла запиту;

deny — блокує запит;

log — записує інцидент у журнал;

msg — повідомлення для системного журналу.

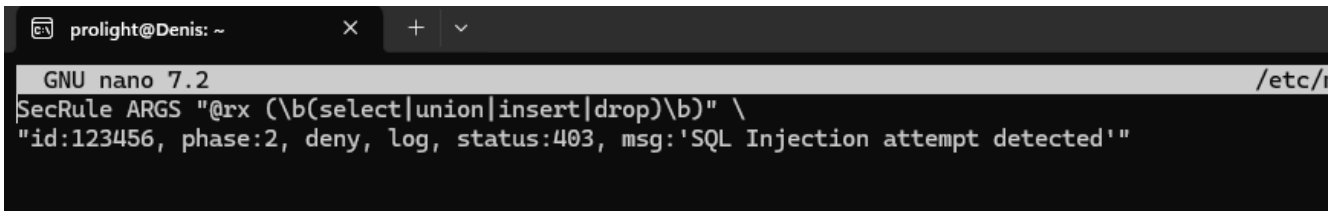

Рис. 3.10 – Власне правило для виявлення SQL-ін’єкцій

Для того, щоб правило було активним, воно було підключене у файл конфігурації Apache:

Include /etc/modsecurity/custom-rules.conf

Файл /etc/apache2/mods-enabled/security2.conf містить усі активовані правила. Після додавання власного правила, конфігурація була перевірена командою:

sudo apachectl configtest

і після отримання повідомлення "Syntax OK" сервер був перезапущений:

sudo systemctl restart apache2

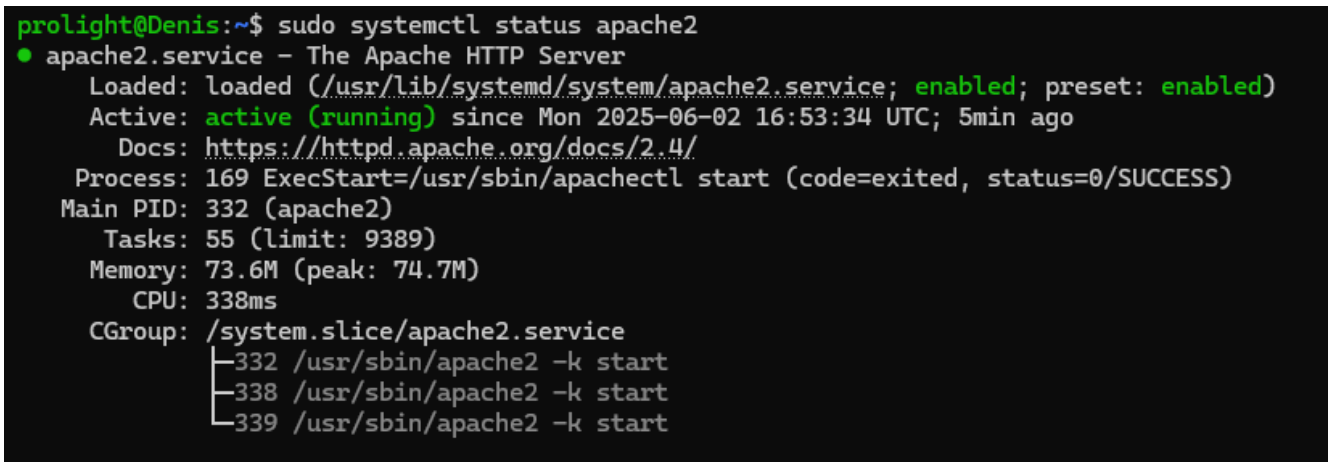

Рис. 3.11 – Запуск Apache після додавання власного правила

Імітація атаки SQL-ін'єкції

Для перевірки роботи системи був здійснений запит з типовою SQL-ін'єкцією у параметрі id:

http://localhost/?id=1+union+select+1,2,3

У результаті сервер повернув HTTP 403 Forbidden, що свідчить про успішне блокування запиту.

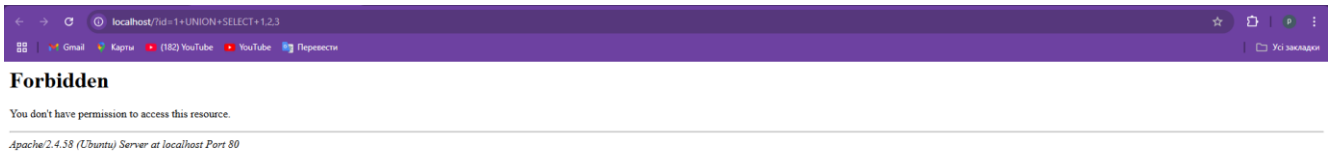

Рис. 3.12 – Блокування SQL-ін'єкції у відповідь на GET-запит

Аналіз логів у файлі audit.log

Усі спрацювання фіксуються у файлі журналу:

/var/log/apache2/modsec_audit.log

Був виявлений запис, який свідчить про спрацювання як користувацького правила, так і правил з OWASP CRS:

Фрагмент журналу (спрощено):

Message: Access denied with code 403 (phase 2). Matched Data: union select found within ARGS:id: 1 union select [file "/etc/modsecurity/custom-rules.conf"] [line "1"] [id "9999999"] [msg "Custom SQL Injection Rule Triggered"]

Рис. 3.13 – Лог журналу про заблоковану атаку за власним правилом

Крім того, в тому самому лог-файлі фіксується спрацювання правил з OWASP CRS (наприклад, ID 942100 — SQLi через libinjection, ID 949110 — перевищення порогу аномалії):

Message: SQL Injection Attack Detected via libinjection [file "/usr/share/modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"]
[id "942100"]

Message: Inbound Anomaly Score Exceeded (Total Score: 15)
[file "/usr/share/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"]
[id "949110"]

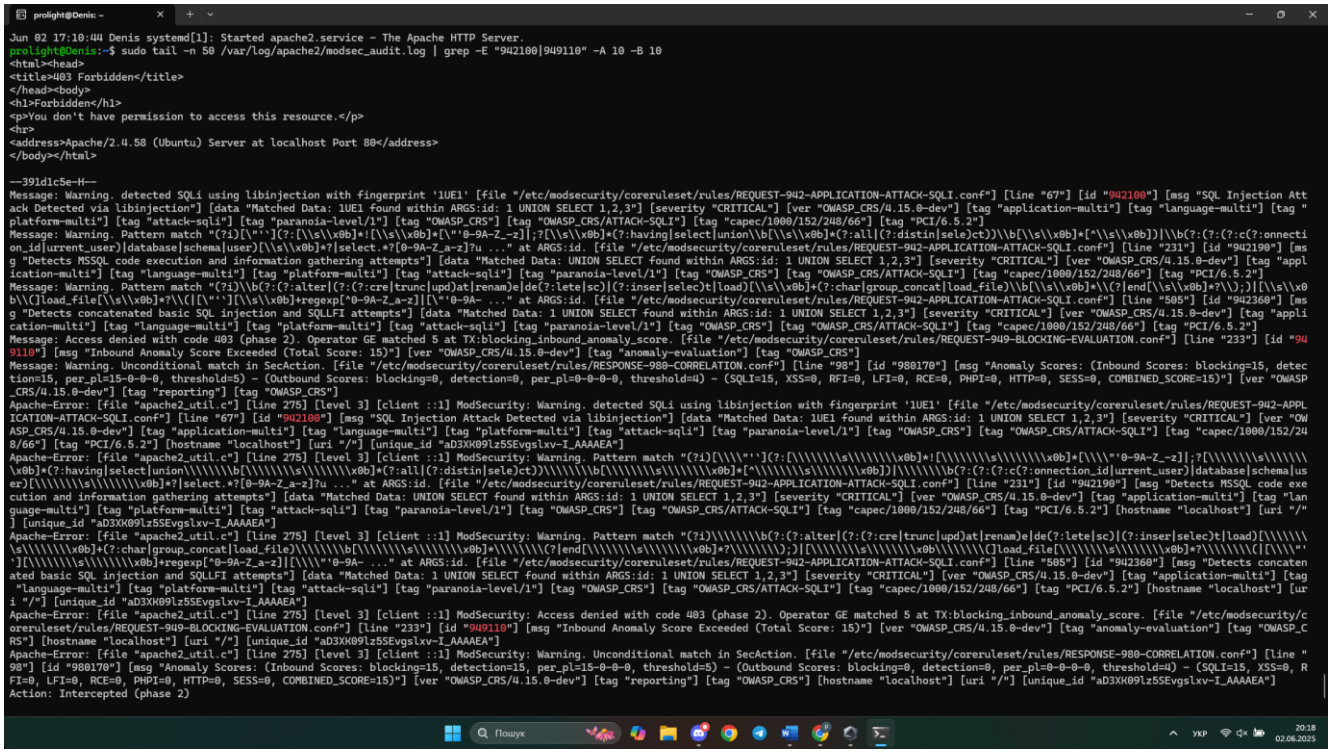

Рис. 3.14 – Спрацювання правил OWASP CRS на SQL-ін’єкцію

Порівняння ефективності CRS та користувацького правила

Як видно з логів, одночасно спрацювало декілька правил:

користувацьке правило з id 9999999,

сигнатурне правило OWASP CRS (942190),

правило виявлення через libinjection (942100),

правило блокування за оцінкою аномалії (949110).

Це свідчить про комплексність захисту, яка забезпечується спільним використанням:

власних правил для специфічних запитів у проєкті;

бібліотеки libinjection для швидкого виявлення шаблонів SQL;

OWASP CRS для широкого спектру відомих атак.

3.3 Рекомендації щодо підвищення захисту з використанням MoSecurity

Розгортання модуля ModSecurity для веб-сервера Apache є ефективним кроком у підвищенні кіберстійкості вебдодатків та інформаційних ресурсів загалом. Зокрема, під час виконання експериментальної частини дипломного дослідження було продемонстровано успішне виявлення та блокування атак типу SQL-ін’єкція як на основі стандартних правил OWASP Core Rule Set, так і з використанням користувацьких сигнатур.

Однак, як свідчить практика, налаштування системи безпеки лише одного модуля — це лише початковий етап. Для досягнення повної ефективності системи захисту важливо постійно її розвивати, адаптувати до нових загроз, оновлювати й інтегрувати з іншими компонентами безпекової інфраструктури.

У цьому підрозділі запропоновано комплекс рекомендацій, що охоплюють технічні, організаційні та процедурні аспекти вдосконалення системи захисту вебресурсу від SQL-ін’єкцій, яка реалізована з використанням модуля ModSecurity.

Підвищення глибини фільтрації за допомогою розширених правил

Базова конфігурація OWASP Core Rule Set (CRS), яка була активована під час виконання практичних завдань, є ефективною для виявлення найбільш поширених типів SQL-ін’єкцій. Однак її можна розширити для кращого охоплення складних атак за допомогою:

активації додаткових рівнів параної (paranoia levels):
— PL1 (типово) виявляє базові SQLi, XSS, LFI;
— PL2 включає глибшу перевірку синтаксису та шаблонів ін’єкцій;
— PL3 і PL4 рекомендуються для високочутливих додатків, але можуть спричинити помилкові спрацювання.

створення користувацьких правил з урахуванням особливостей логіки вебдодатку. Наприклад, якщо відомо, що параметр id повинен містити лише цілі числа, можна задати правило:

SecRule ARGS:id "!^\d+$" "id:100100,phase:2,deny,status:403,msg:'Non-numeric id parameter detected'"

додавання правил на основі поведінкової евристики (наприклад, багаторазові спроби запитів з різними структурами в короткий проміжок часу).

Актуалізація бази правил і захист від 0-day атак

Оскільки загрози постійно змінюються, зловмисники розробляють нові способи обходу фільтрів, тому слід:

Регулярно оновлювати версію OWASP CRS (рекомендовано відстежувати офіційне сховище GitHub: https://github.com/coreruleset/coreruleset);

Перевіряти наявність патчів для самого ModSecurity (особливо якщо використовується версія 2.x, що не підтримує всі новітні функції);

Налаштувати систему автоматичного моніторингу безпекових оновлень (apt-listchanges, unattended-upgrades тощо).

Окрім того, рекомендується враховувати поширені методи обфускації SQL-запитів (наприклад, чергування регістрів, кодовані символи, вставки у вигляді коментарів) та оновлювати сигнатури, щоб покривати ці техніки.

Централізоване ведення журналів та аналітика

Журнал audit.log, який створюється ModSecurity, містить важливу інформацію про запити, що були заблоковані або позначені як потенційно небезпечні. Для того щоб ця інформація була максимально корисною, потрібно:

Збирати журнали на централізованому сервері журналювання;

Налаштувати ротацію логів (logrotate) для запобігання переповненню диску;

Інтегрувати з інструментами аналітики (наприклад, ELK Stack: Elasticsearch, Logstash, Kibana);

використовувати Dashboards для швидкої візуалізації спрацювань за ID правил, джерелами IP, типами запитів.

Додатково можливо реалізувати обробку логів за допомогою утиліт: grep, awk, jq — для витягування ключових параметрів запитів.

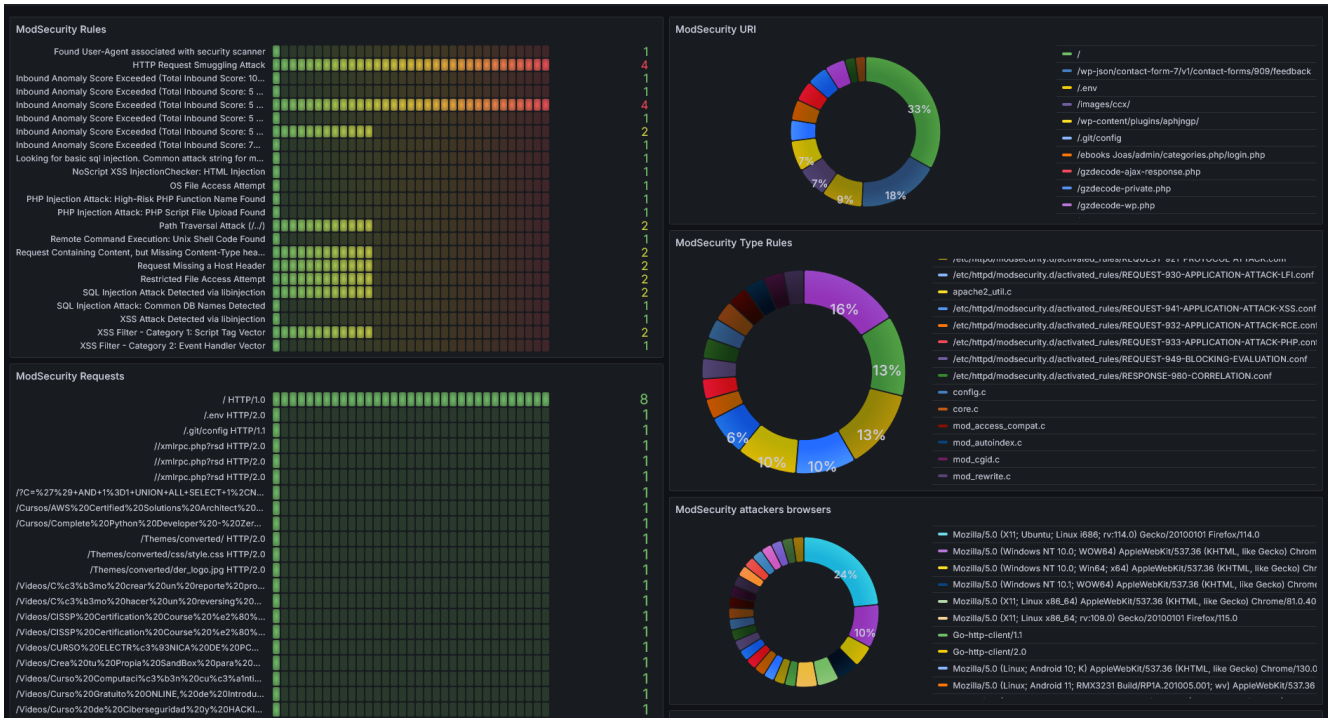

Рис. 3.15 – Візуалізація журналів ModSecurity у Kibana

Інтеграція з SIEM-системами

SIEM (Security Information and Event Management) дозволяє отримувати повну картину стану безпеки інформаційного ресурсу, виявляти аномалії, корелювати події з різних джерел.

Інтеграція ModSecurity з SIEM дозволяє:

Об’єднати журнали з вебсервера, IDS, антивірусів, операційної системи;

Налаштувати автоматичне реагування на виявлені SQLi;

Створити сповіщення (alerts) у разі підозрілої активності;

Підвищити відповідність стандартам безпеки (ISO/IEC 27001, PCI DSS).

Рекомендовані SIEM-рішення: OSSIM, Wazuh, Splunk, QRadar (для великих організацій).

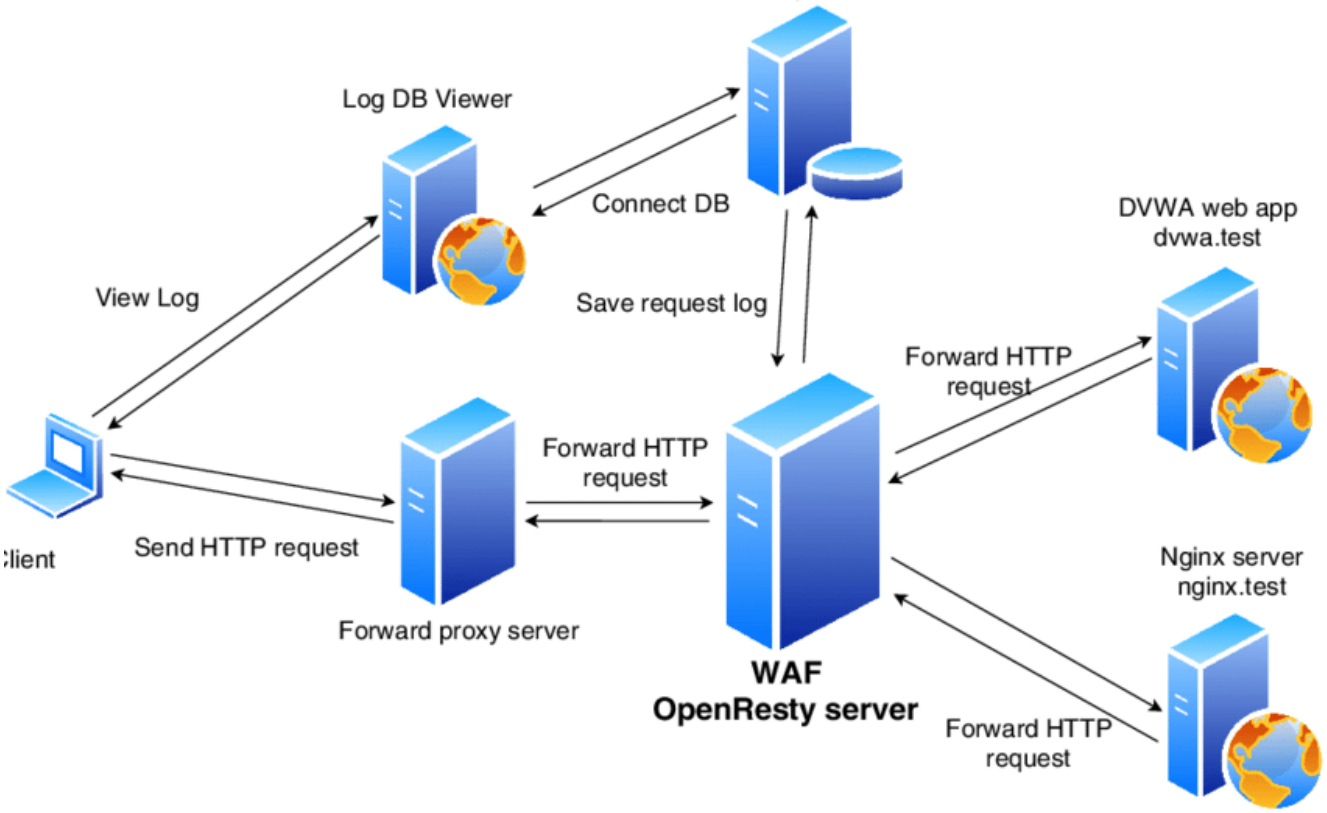

Рис. 3.16 – Інтеграція ModSecurity із системою моніторингу подій

Валідація вхідних даних на прикладному рівні

Жоден WAF не замінює внутрішню обробку даних у додатку. Тому важливо:

використовувати параметризовані запити (prepared statements) у коді бекенду;

Не допускати складання SQL-запитів шляхом конкатенації з вхідними параметрами;

Обмежувати типи й формат вхідних значень (довжина, перелік допустимих символів).

Усі вхідні дані мають валідуватися як на клієнтському, так і на серверному боці.

Регулярне тестування системи захисту

Перевірка системи захисту — ключовий етап, без якого неможливо переконатися в її ефективності. Рекомендується:

Використовувати sqlmap для симуляції SQL-ін’єкцій: sqlmap -u "http://localhost/?id=1" --batch --risk=3 --level=5

Проводити періодичні сканування через Burp Suite та OWASP ZAP;

Створити власний тестовий набір запитів з різними варіантами SQL-ін’єкцій;

Оформити результати перевірок у вигляді звіту з рекомендаціями.

Якщо в системі з часом з’являться нові параметри, поля, API-ендпоїнти — необхідно оновити як WAF-правила, так і тестовий набір.

Резервне копіювання конфігурацій і журналів

Для запобігання втраті важливих налаштувань або історії спрацювань, слід:

Налаштувати автоматичне резервне копіювання файлів /etc/modsecurity/*.conf;

Копіювати журнали на віддалений сервер;

Використовувати інструменти для інкрементального бекапу (rsync, borgbackup);

Зберігати контрольні хеші (SHA256) конфігурацій для виявлення несанкціонованих змін.

Виявлення ботів і автоматичних сканерів

Часто SQL-ін’єкції надсилаються автоматизованими сканерами. Для зменшення навантаження та запобігання атакам слід:

Створити правила на основі User-Agent (наприклад, "sqlmap", "python-requests");

Встановити обмеження на частоту запитів з однієї IP-адреси;

Інтегрувати ModSecurity з Fail2ban або iptables для тимчасового блокування.

Додаткові рівні захисту

Система захисту може бути посилена такими компонентами:

Cloud-based WAF (Cloudflare, AWS WAF) — додатковий захист ще до досягнення вебсервера;

Honeypot – для виявлення автоматизованих сканерів і ботів;

CAPTCHA або JS-челенджі – проти масових запитів.

Також слід обмежити доступ до службових частин сайту (наприклад, /admin, /phpmyadmin), застосувавши автентифікацію, IP-фільтрацію або VPN-доступ.

Підготовка до сертифікацій та відповідність стандартам

Якщо вебдодаток обробляє критичні дані (персональні, платіжні), важливо:

Впровадити політики безпеки відповідно до вимог GDPR, ISO/IEC 27001, PCI DSS;

Оформити документацію щодо WAF і правил безпеки;

Зберігати лог аудиту всіх змін системи захисту.

Це дозволить як підтвердити відповідність вимогам, так і спростити проходження сертифікацій та аудитів.

ВИСНОВОК

У межах виконаної дипломної роботи було досліджено одну з найбільш поширених та небезпечних загроз для вебзастосунків — SQL-ін’єкції, а також розроблено і впроваджено модуль захисту інформаційного ресурсу від даного типу атак з використанням модуля ModSecurity. Актуальність теми обумовлена широким поширенням вебдодатків у різних сферах діяльності та зростаючою кількістю цілеспрямованих атак, спрямованих на компрометацію баз даних через маніпуляції SQL-запитами.

На початковому етапі роботи було здійснено теоретичний аналіз проблеми. Було досліджено механізми SQL-ін’єкцій, зокрема класичні (in-band), ін’єкції сліпого типу (blind), time-based ін’єкції та union-based запити. Розглянуто методи впровадження шкідливих SQL-конструкцій у параметри запитів (GET/POST), а також способи обфускації запитів для обходу базових фільтрів. Було встановлено, що SQL-ін’єкція залишається однією з п’яти найнебезпечніших вебвразливостей згідно з OWASP Top 10, та може призвести до витоку конфіденційних даних, знищення бази даних, ескалації привілеїв, захоплення адміністрування сайту тощо.

Далі було проаналізовано сучасні засоби виявлення та запобігання SQL-ін’єкціям, зокрема: застосування параметризованих запитів (prepared statements), серверну валідацію даних, фільтрацію на стороні клієнта, використання проксі та систем виявлення вторгнень. Особливу увагу приділено фаєрволу вебзастосунків (WAF) як одному з найефективніших рішень для раннього виявлення атак. У цьому контексті модуль ModSecurity було обрано як основу для побудови практичного захисту через його гнучкість, відкритий вихідний код, сумісність із Apache та підтримку правилового набору OWASP Core Rule Set.

У третьому розділі дипломної роботи реалізовано розгортання та налаштування модуля ModSecurity на вебсервері Apache у середовищі Ubuntu. Описано всі етапи: встановлення, увімкнення модуля, копіювання та редагування основного конфігураційного файлу, перевірка синтаксису та перезапуск сервера. Проведено імпорт базового набору правил OWASP CRS, який забезпечує виявлення поширених типів атак, зокрема SQLi, XSS, LFI, RFI та інші.

Особливу увагу приділено налаштуванню користувацького правила для виявлення запитів, які містять підозрілі SQL-конструкції (наприклад, select, union, insert, drop). Запропоноване правило успішно спрацювало під час тестових SQL-ін’єкцій, що підтверджено через HTTP-відповідь 403 Forbidden та відповідний запис у журналі audit.log. Крім того, було зафіксовано спрацювання вбудованих правил OWASP CRS з ID 942100 (SQLi через libinjection), 942190 та 949110 (перевищення аномалійного порогу).

У результаті практичної реалізації було продемонстровано ефективність комплексного підходу до захисту: як стандартні правила, так і власні сигнатури спрацювали у відповідь на типові SQL-атаки. Блокування відбулося ще на рівні аналізу HTTP-запиту, до того, як дані потрапили до бази. Це свідчить про здатність системи захистити застосунок у реальному часі навіть без участі бекенд-логіки.

Крім того, в роботі було запропоновано низку практичних рекомендацій щодо вдосконалення системи захисту:

Регулярне оновлення правил OWASP CRS та перехід на актуальну версію (наприклад, 4.0 або вище);

Активація додаткових рівнів «параної» для глибшого аналізу (PL2, PL3);

Інтеграція з системами журналювання (наприклад, ELK Stack або Wazuh) для централізованої обробки логів;

Інтеграція з SIEM-системами для кореляції подій і створення алертів;

періодичне тестування системи захисту за допомогою sqlmap, OWASP ZAP, Burp Suite;

резервне копіювання конфігурацій та логів;

створення honeypot-зон для виявлення активності ботів і автоматизованих сканерів.

Усі ці кроки спрямовані на побудову багаторівневої, гнучкої та масштабованої архітектури захисту, яка здатна адаптуватися до змін середовища загроз. Реалізований у рамках дипломної роботи модуль ModSecurity може бути використаний як компонент комплексної системи безпеки в умовах реального корпоративного середовища.

Таким чином, мету дипломної роботи досягнуто: досліджено проблему SQL-ін’єкцій, реалізовано на практиці систему захисту з використанням сучасного модуля, здійснено аналіз ефективності спрацювання правил та запропоновано комплекс рекомендацій щодо подальшого розвитку захисту вебресурсів.

Запропоноване рішення може бути рекомендоване до впровадження в малих і середніх організаціях, де є потреба у підвищенні рівня безпеки вебдодатків без значних витрат на комерційні продукти. Крім того, результати дослідження можуть бути використані як основа для подальших наукових і практичних розробок у сфері кібербезпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1 OWASP (Open Web Application Security Project) – https://owasp.org
2 ModSecurity – https://modsecurity.org
3 OWASP Core Rule Set – https://coreruleset.org
4 SQLMap – https://sqlmap.org
5 Acunetix Web Vulnerability Scanner – https://www.acunetix.com
6 Apache HTTP Server Project – https://httpd.apache.org
7 Ubuntu Server Documentation – https://ubuntu.com/server/docs
8 Python SQLAlchemy ORM – https://www.sqlalchemy.org
9 Django Web Framework – https://www.djangoproject.com
10 Laravel PHP Framework (Eloquent ORM) – https://laravel.com
11 Hibernate ORM – https://hibernate.org
12 Entity Framework – https://learn.microsoft.com/en-us/ef/
13 OWASP ZAP Proxy – https://www.zaproxy.org
14 WAFNinja – https://github.com/PortSwigger/wafninja
15 Snort IDS – https://www.snort.org
16 Suricata IDS/IPS – https://suricata.io
17 ModSecurity GitHub Repository – https://github.com/SpiderLabs/ModSecurity
18 Core Rule Set GitHub Repository – https://github.com/coreruleset/coreruleset
19 Cerberus (Python validation library) – https://docs.python-cerberus.org
20 Pydantic – https://docs.pydantic.dev
21 RegExp (MDN Web Docs) – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions
22 Apache ModSecurity Audit Logs – https://github.com/SpiderLabs/ModSecurity/wiki/ModSecurity-2-Data-Formats

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)