КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (28), 2025
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2025.28.756
УДК 004.056.53, 004.056.5

Маркевич Михайло Володимирович
студент кафедри безпеки інформаційних технологій
Національний Університет «Львівська Політехніка», Львів, Україна
ORCID ID: 0009-0000-3228-3525
mykhailo.markevych.mkbbi.2024@lpnu.ua

Горячий Олег Ярославович
асистент кафедри безпеки інформаційних технологій
Національний Університет «Львівська Політехніка», Львів, Україна
ORCID ID: 0000-0003-4948-458X
oleh.y.horiachyi@lpnu.ua

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СЕРВЕРНИХ АТАК НА РЕЛЯЦІЙНІ ТА НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ. СТВОРЕННЯ СТРАТЕГІЇ ЗАХИСТУ

Анотація. Сучасний стан розвитку інформаційних технологій характеризується широким використанням баз даних у різних сферах діяльності, зокрема, в бізнесі, медицині, науці та державному управлінні. Зростання обсягів даних та їх цінності призводить до збільшення кількості кібератак на бази даних. У зв'язку з цим питання забезпечення безпеки баз даних набуває особливої актуальності. У статті розглянуто ефективність серверних атак на реляційні та нереляційні бази даних. Проведено детальний аналіз методів здійснення атак, таких як SQL/NoSQL ін'єкції та словникові атаки, а також оцінено їхні наслідки для безпеки інформаційних систем. Проаналізовано ефективність перебору паролів із зафіксованим словником залежно від параметрів хеш-функцій, кількості раундів хешування бібліотеки bcrypt. Показано, що зі збільшенням кількості раундів хешування обчислювальна стійкість підбору пароля зростає, тому кожна спроба займає більше часу для обробки. Однак, навіть з обмеженим словником та ефективним методом перебору, атаку можна виконати за досить короткий проміжок часу, якщо параметри хешування вибрані неправильно. Наведені схематичні рисунки атак на відповідні типи баз даних. Запропоновано комплексну стратегію захисту по забезпеченню конфіденційності, цілісності та доступності інформації, що включає в себе попередню обробку даних користувачів, хешування паролів, встановлення лімітів на кількість запитів, розмежування доступу та блокування комп'ютеризованих дій. Описано та реалізовано методи протидії серверним атакам, зокрема розглянуто функціональні бібліотеки для безпечного зберігання паролів користувачів. Крім цього, проведено порівняння ефективності різних типів атак у контексті існуючих методів захисту. Результати дослідження продемонстрували, що комплексне впровадження базових компонетів стратегії захисту суттєво посилює стійкість даних до типових серверних атак, особливо в умовах масштабованого середовища із великою кількістю точок входу.

Ключові слова: бази даних; інформаційна безпека; стратегія захисту; NoSQL-ін'єкції; керування доступом на основі ролей; словникова атака; комбінована атака; SQL-ін'єкції.

ВСТУП

З розвитком технологій та збільшенням кількості пристроїв, підключених до мережі, актуальність проблеми захисту інформації, що зберігається в базах даних, також зростає. Сучасні веб-застосунки використовують як реляційні, так і нереляційні бази даних (БД) для зберігання та обробки великих обсягів даних [1]. Однак такі БД стають мішенню для різноманітних серверних атак, зокрема SQL- та NoSQL-ін'єкцій, а також

словникових атак, що можуть призвести до витоку конфіденційної інформації, порушення цілісності даних і збоїв у роботі системи [2], [3]. Розробленння комплексної стратегії захисту БД, що включає: вдосконалені методи обробки даних, хешування паролів, встановлення лімітів на запити та використання CAPTCHA, дозволить підвищити рівень безпеки.

Об'єктом дослідження в цій статті є веб-застосунок, що використовує реляційні та нереляційні БД для зберігання й обробки інформації.

Предметом дослідження виступають серверні атаки на реляційні та нереляційні БД, такі як: SQL/NoSQL-ін'єкції та словникові атаки, а також стратегія захисту, елементи якої наведено вище.

Постановка проблеми. Як уже зазначено у вступі, розширення технологій спричинило збільшення обсягів даних, що зберігаються й обробляються в БД. Впровадження SQL є загрозою для реляційних БД, оскільки зловмисники можуть використовувати маніпуляції зі структурованими запитами для отримання або модифікації конфіденційної інформації. Основною загрозою для нереляційних БД (NoSQL) є NoSQL-ін'єкції, які експлуатують відсутність жорсткої типізації запитів у поєднанні з недосконалими механізмами автентифікації. Це може призвести до витоку даних, що, у свою чергу, може зупинити бізнес-процеси в організаціях або стати загрозою національної безпеки. Дослідження ефективності таких атак дозволить виявити найбільш суттєві вразливості та обрати оптимальну стратегію захисту своїх внутрішніх активів.

Аналіз останніх досліджень і публікацій. Реляційні та нереляційні БД — два основних типи систем управління базами даних (СУБД), що використовуються для зберігання та організації інформації. Кожен із цих типів має свої унікальні особливості та характеристики з точки зору захисту інформації та її обробки [1]. Реляційні БД відомі своєю структурованістю, вони використовуються для зберігання інформації та забезпечення доступу до неї. Нереляційні БД, навпаки, використовують інші підходи до організації даних і доступу до них [2].

Останні дослідження в області безпеки БД демонструють зростання загрози атак на СУБД. У статті [4] розглядаються сучасні тенденції розвитку NoSQL баз даних, зокрема аспекти їх безпеки та ризики, пов'язані з відкритим доступом до таких систем. Автори наголошують на необхідності вдосконалення механізмів автентифікації та контролю доступу для запобігання несанкціонованому використанню NoSQL сховищ.

У дослідженні [5] проаналізовано вразливості MongoDB до атак ін'єкційного типу та запропоновано методи їх запобігання. Автори демонструють ефективність різних підходів до захисту, таких як перевірка запитів і використання механізмів обмеження доступу. Водночас у статті [6] розглянуто поширеність атак на SQL бази даних, зокрема SQL-ін'єкції та їх наслідки. Запропоновано комплексний підхід до запобігання таким атакам, що включає шифрування запитів, використання параметризованих операторів SQL і багаторівневу автентифікацію.

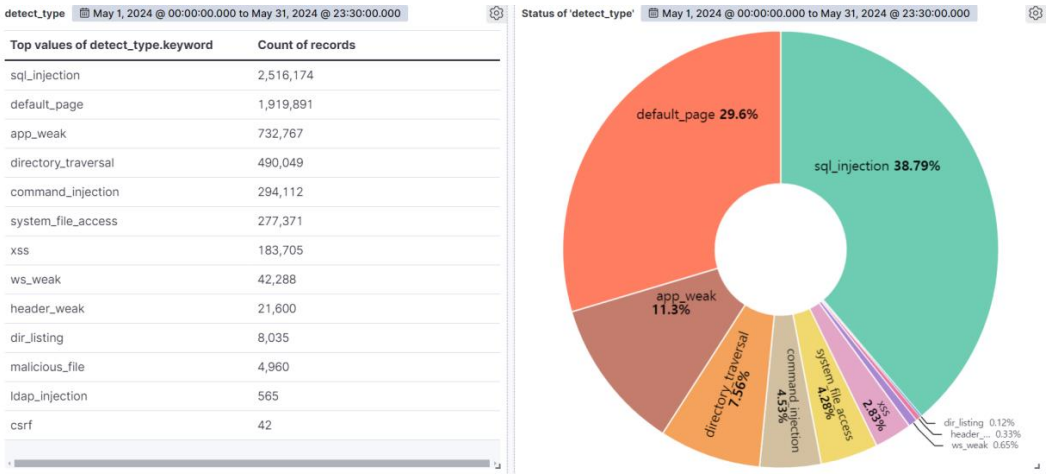
Мета статті. Метою даної статті є дослідження ефективності різних серверних атак на реляційні та нереляційні БД, а також розробка стратегії їх захисту. **Завдання** полягає в аналізі сучасних методів захисту БД і впровадженні комплексної стратегії, яка включатиме вдосконалені методи обробки даних, хешування паролів, обмеження кількості запитів і перевірку через капчу. Також необхідно змоделювати комбіновану атаку на БД зловмисником та оцінити ефективність впровадження захисних механізмів у веб-застосунку. Механізми захисту реалізвано на серверній частині за допомогою Node.js та на клієнтській частині за допомогою React.js.

ISSN 2663 - 4023

ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

Огляд атак на бази даних

Серверні атаки на БД — це цілеспрямовані дії зловмисників, спрямовані на отримання несанкціонованого доступу до даних, вплив на їхню цілісність або блокування доступу до БД. Для дослідження серверних атак необхідно проаналізувати статистичні дані про найбільш поширені з них. З веб-порталу monitorapp.com [7] можна побачити інформацію щодо поширення вразливостей веб-застосунків у відсотковому відношенні. Цей рейтинг, як не дивно, очолює «SQL Injection» (рис. 1). Саме ця атака буде розглянута більш детально в цій статті.



Top values of detect_type.keyword	Count of records
sql_injection	2,516,174
default_page	1,919,891
app_weak	732,767
directory_traversal	490,049
command_injection	294,112
system_file_access	277,371
xss	183,705
ws_weak	42,288
header_weak	21,600
dir_listing	8,035
malicious_file	4,960
ldap_injection	565
csrf	42

Рис. 1. Демонстрація найпоширеніших вразливостей за травень 2024 року

SQL-ін'єкція (SQLI) — це одна з найпоширеніших атак, при якій зловмисник використовує SQL запити для видалення, модифікації або отримання конфіденційної інформації з БД [8]. Наслідки включають втрату конфіденційної інформації, порушення цілісності даних та навіть втрату контролю над БД.

NoSQL-ін'єкція (NoSQLI) — форма атаки на безпеку даних, що виникає при використанні NoSQL баз даних. Під час таких атак зловмисник використовує неправильну обробку введених даних, намагаючись змінити логіку запиту до БД. Це може призвести до серйозних наслідків, включаючи несанкціонований доступ до конфіденційної інформації або навіть втрату даних.

З відомого веб-порталу comparitech.com [9] можна також побачити статистичну інформація про використання однакових паролів у різних веб-застосунках (рис. 2).

Рис. 2. Демонстрація статистики використання однакових паролів

ISSN 2663 - 4023

З цієї статистичної діаграми видно, що лише 35% респондентів використовують різні паролі для різних сайтів та облікових записів. Водночас 52% респондентів не завжди, але інколи можуть використати той самий пароль на декількох облікових записах (не всіх). І 13% респондентів взагалі використовують однакові паролі всюди.

Словникова атака (dictionary attack) — це форма атаки грубої сили, під час якої хакери намагаються вгадати дані для автентифікації користувачів, зокрема паролі. Для цього вони використовують спеціалізоване програмне забезпечення, щоб швидко перевірити список часто вживаних слів, фраз та цифрових комбінацій. Якщо атака успішна, то зловмисник може отримати доступ до різних облікових записів, зокрема банківських рахунків, профілів у соціальних мережах та захищених паролем файлів.

У контексті серверних атак можна розглядати комбіновані атаки, які використовують різні методи для досягнення своєї мети. Наприклад, атака може починатися з фішингового листа, мета якого — змусити користувача натиснути на посилання або надати свої особисті дані. Якщо користувач реагує на лист, відправляє свої облікові дані або завантажує невідому програму, яка може бути троянським конем або майнером, зловмисник отримує доступ до особистих даних, обчислювальних або фінансових ресурсів жертви.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Приклади застосування різних типів SQL-ін’єкцій

Як було описано раніше, SQLI є вразливістю, яка дозволяє зловмисникам виконувати SQL запити для отримання несанкціонованого доступу до даних. Зазначимо, що в цьому розділі даними для атаки буде слугувати інформація про користувачів. Нижче подано схему, яка ілюструє принцип роботи атаки через реляційну базу MySQL (рис. 3).

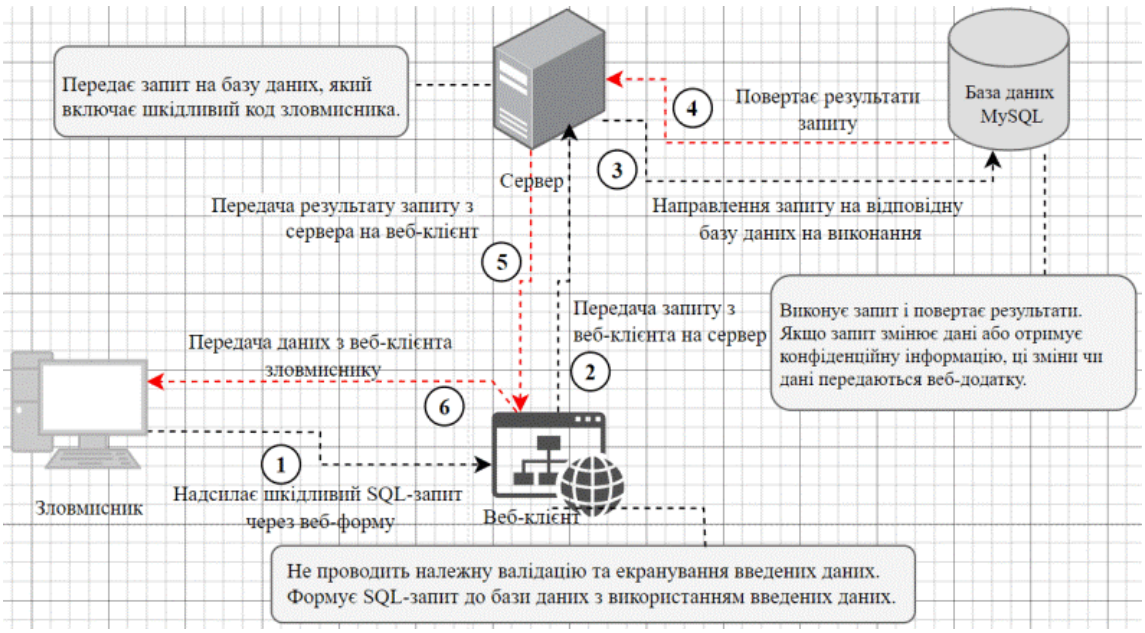


Рис. 3. Ілюстрація принципу дії SQL-ін’єкції

ISSN 2663 - 4023

Описуючи атаку SQLI, слід розуміти, що вона має кілька підвидів, тобто типів атак. Деякі з них можуть мати більш критичні наслідки для скомпрометованої системи в разі успіху. Нижче наведена класифікація цієї атаки (рис. 4).

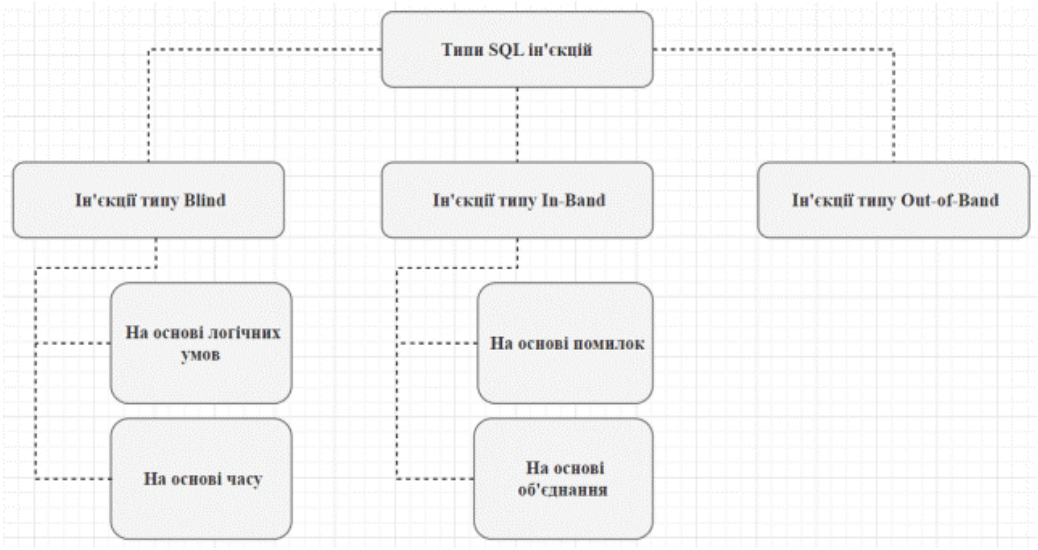


Рис. 4. Класифікація різних типів SQL-ін'єкцій

Blind ін'єкції дещо відрізняються від інших методів SQLI, оскільки зловмисник не бачить безпосередньо результату запиту [10]. Натомість він контролює поведінку сервера або час, необхідний для отримання відповіді. Коли зловмисник вводить умови, які змінюють поведінку сервера, це називається **умовним впровадженням**. Наприклад, якщо умова істинна, сервер надає одну сторінку, а якщо умова хибна — іншу. Це дозволяє зловмиснику дізнатися деякі деталі про БД. Нижче наведено приклад запиту, який може надійти від зловмисника:

```
SELECT * FROM users WHERE id = 1 AND 1=1         -- (істинно)
SELECT * FROM users WHERE id = 1 AND 1=2         -- (хибно)
```

Крім того, існує ін'єкція **на основі часу**, яка вводить команди затримки в БД, щоб визначити, чи запит є істинним. Зловмисник надсилає SQL-код з наміром змусити БД чекати певну кількість секунд перед тим, як сервер зможе відповісти. Затримка буде свідчити про те, що запит виконано. Приклад для цього типу SQLI:

```
SELECT * FROM users WHERE id = 1; IF 1=1 WAITFOR DELAY '00:00:35'--
```

In-Band тип є одним із найпоширеніших типів SQLI, який полягає в використанні одного каналу для атаки та витоку даних. Одним з підтипів цього типу ін'єкцій є ін'єкція **на основі помилок**. Зловмисник отримує повідомлення про помилку, щоб здобути інформацію про структуру БД сервера, що дозволяє йому отримати доступ до даних. Для цього зловмисник вставляє власний SQL запит в оператор, який генерує помилку. З цього повідомлення можна отримати важливу інформацію, таку як імена таблиць та стовпців, якщо його належним чином розібрати. Прикладом такого запиту може бути наступний:

```
SELECT * FROM users WHERE id = 1' AND 1=CONVERT(int,(SELECT @@version))--
```

ISSN 2663 - 4023

Ще одним підтипом є ін'єкція **на основі об'єднання**. Цей метод передбачає використання оператора UNION, який поєднує результати двох різних запитів в один. Зловмисник вводить послідовність SQL-інструкцій і, зрештою, використовує оператор UNION, щоб об'єднати вихідні дані другого запиту з першим, тим самим отримуючи доступ до інформації з БД. Приклад такого запиту наведено нижче:

```
SELECT name, surname FROM users WHERE id = 1 UNION SELECT username,
password FROM admins--
```

Out-of-band ін'єкції не завжди можливі, а здійсненні тільки тоді, коли на сервері БД активовано певні функції. Цей тип ін'єкції використовується, коли зловмисник не може застосувати той самий канал як для атаки, так і для збору даних, або коли сервер працює надто повільно чи нестабільно. У цьому випадку зловмисник отримує інформацію з БД через виконання зовнішніх запитів, таких як DNS або HTTP. Цей метод дозволяє надсилати запит через один протокол (наприклад, HTTP) і отримувати відповіді через інший (наприклад, DNS). У цій роботі даний тип ін'єкцій застосовуватися не буде.

Приклад застосування NoSQL-ін'єкції

NoSQL-ін'єкції стають дедалі актуальнішими через зростання популярності NoSQL баз даних, таких як MongoDB. Як згадано раніше, в даній статті БД налаштовано через модуль Mongoose [11]. Важливою особливістю NoSQL систем є те, що вони зберігають та індексують дані в інших форматах, ніж традиційні реляційні таблиці. Попри цю відмінність, механізм NoSQL-ін'єкцій має певні аналогії з SQL-ін'єкціями, оскільки він також використовується для маніпуляції запитами до БД. Це може призвести до компрометації веб-застосунку та витоку інформації. На рис. 5 представлено ілюстрацію принципу роботи NoSQL-ін'єкції.

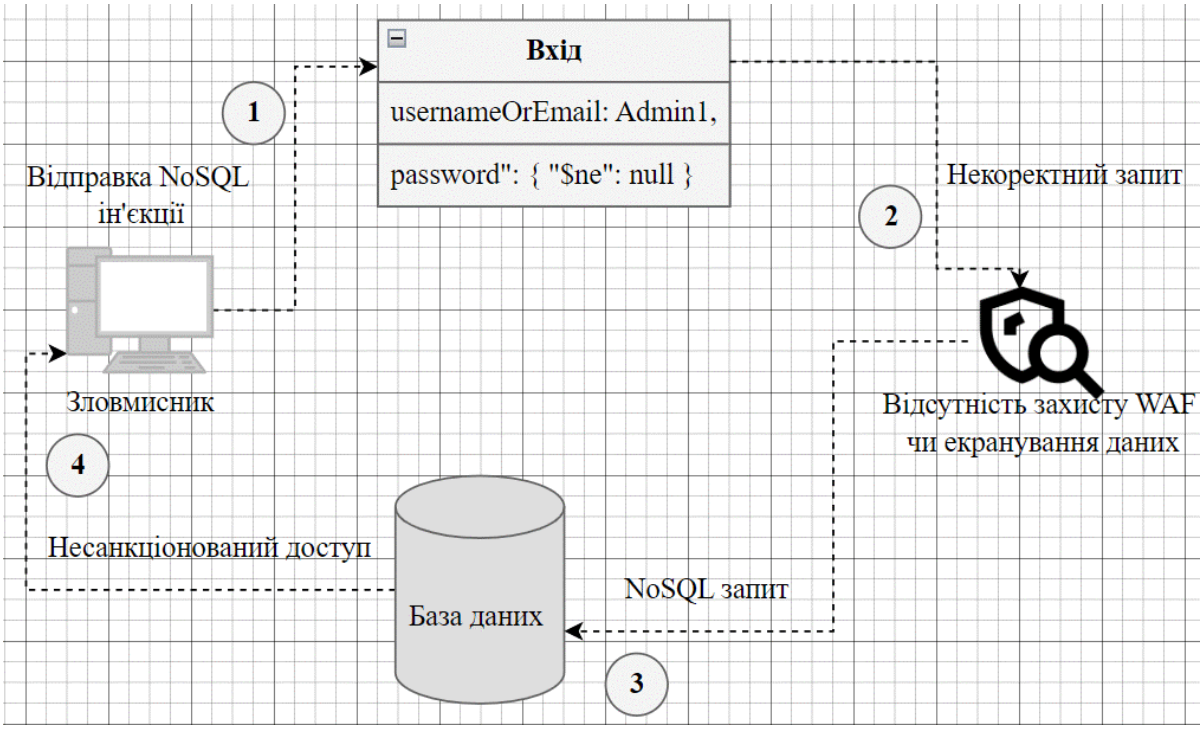


Рис. 5. Ілюстрація принципу роботи NoSQL-ін'єкції на прикладі

Існує два типи атак NoSQLI: операторна та синтаксична [12]. Обидва типи становлять значну загрозу для систем, що піддаються таким атакам. **Операторна** ін'єкція виникає, коли зловмисник використовує спеціальні оператори NoSQL для маніпуляції запитами. Ці оператори змінюють логіку запиту таким чином, щоб отримати несанкціонований доступ до користувацьких даних. Даний тип атаки часто застосовується для обходу механізмів автентифікації. Чудовим прикладом для цієї роботи буде наступна ін'єкція, в якій дані передаються у форматі JSON:

```
{"usernameOrEmail": "Admin", "password": { "$gt": "" }}
```

У цьому випадку використання оператора $gt із порожнім рядком поверне всі записи, де поле password містить будь-яке значення, що задовольняє умові «більше порожнього рядка». **Синтаксичний** тип NoSQL-ін'єкції виникає при навмисному порушенні синтаксису NoSQL-запиту, що дозволяє зловмиснику вставити свій власний шкідливий код. За задумом цей метод доволі схожий на SQLI, де треба змінити структуру запиту до БД, щоб виконати несанкціоновані операції. Наслідком таких атак може бути компрометація облікового запису користувача. У випадку запиту до MongoDB для автентифікації користувача можна використати `"password": { "$ne": null }`, де оператор $ne містить значення null у полі пароля. Запит поверне всі записи, де поле password не дорівнює null, і зловмисник таким чином обійде перевірку автентифікації.

Приклад застосування словникової атаки

Словникова атака може здаватись найменш складною з погляду реалізації, проте її ефективність в разі успіху складно переоцінити. Як було зазначено в розділі про огляд серверних атак на БД, ця атака передбачає використання файлу, що містить тисячі можливих варіацій паролів або логінів (в разі потреби), після чого вони перевіряються для обраного користувача. Нижче можна побачити принцип дії цієї атаки (рис. 6).

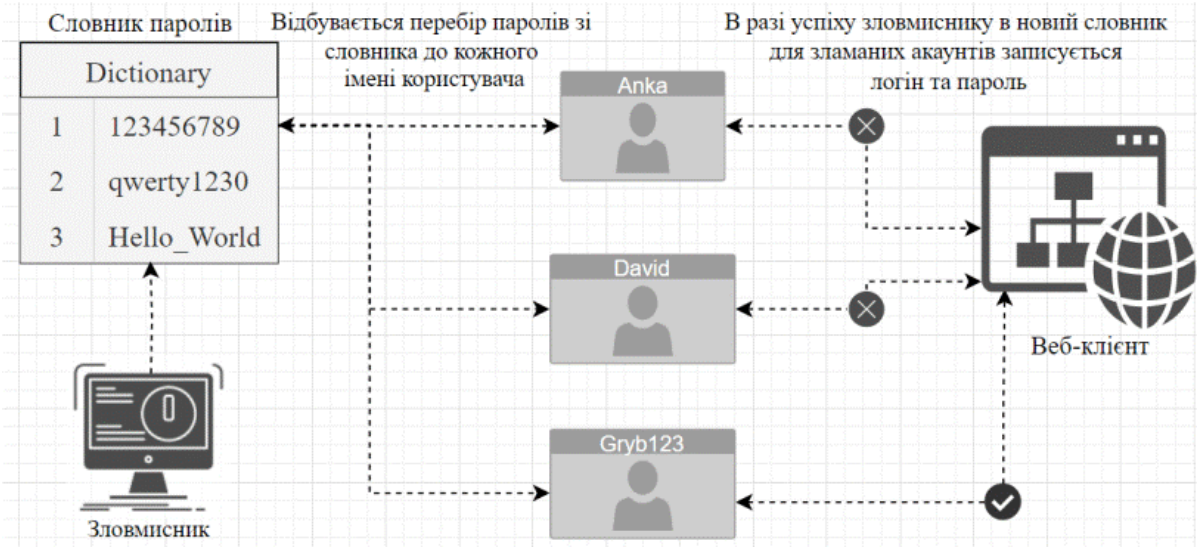


Рис. 6. Ілюстрація принципу дії словникової атаки

Цю атаку також можна скомбінувати з іншою, наприклад, SQLI. Спочатку ми використовуємо вразливість SQLI для отримання імен користувачів платформи та інших даних. Однак серед них нас може зацікавити пароль. Майже завжди паролі зберігаються в

хешованому вигляді, тому не варто розраховувати на пароль у відкритому форматі. Хеш-алгоритм, який використовується, може бути ненадійним, як це було у випадку з SHA-1.

Небезпека полягає в тому, що зловмисник може використати словникову атаку для підбору пароля. Якщо підбір виявиться успішним, він зможе пройти етап автентифікації та отримати доступ, зокрема, до електронної пошти користувача. Після цього зловмисник може спробувати використати скомпрометований пароль для входу на інші ресурси, пов'язані з цією поштою. Можна лише уявити ту катастрофічну шкоду, яку може здійснити зловмисник, якщо скомпрометована основна особиста чи корпоративна пошта. Згідно з ISO 27002 [13], якщо профіль корпоративної пошти працівника було скомпрометовано, жертва повинна негайно повідомити службу підтримки компанії, щоб її доступи були заблоковані на період службового розслідування та ліквідації наслідків.

Ефективність цієї атаки можна буде оцінити, якщо злам буде успішним, пароль підійде та якщо це не займе багато часу. У даній роботі взято словник доволі невеликої довжини — 10000 варіацій паролів [14].

Також словникову атаку можна використати в інший спосіб: коли в результаті SQLI-атаки отримуємо хешовані паролі, можна спробувати використати ту ж хеш-функцію, щоб хешувати паролі словника і перебрати їх, щоб знайти збіг хешів.

У документації по bcrypt зазначено, що модуль використовує різну кількість раундів хешування, що ускладнює перебір паролів. Збільшення кількості раундів призводить до зниження швидкості обчислення хешів. Однак, коли кількість раундів досягає певного рівня, швидкість обчислення стає настільки повільною, що стає важко виміряти її в хешах за секунду, і вона оцінюється в секундах на один хеш.

Реалізація словникової атаки, що розглядається в цій роботі, перебирає всі *10000 паролів* приблизно за *18.3 хвилин*.

Однак, коли є лише хешований пароль, то в разі, якщо відомо, яким способом його хешували, можна оцінити час, необхідний для атаки.

У випадку з об'єктом дослідження — веб-застосунком про фільми — буде недоцільно використовувати хешування, де кількість його раундів перевищує 15. Саме тому з'являється можливість розрахувати, скільки часу знадобиться для перебору всіх 10000 паролів, перевіряючи їх усіма раундами хешування.

Нижче показано, скільки часу займає перевірка паролів зі словника методом *bcrypt.compare бібліотеки bcrypt* та *часом перебору паролів* (табл. 1).

Таблиця 1

Порівняння часу перевірки хешів для різної кількості раундів

Кількість раундів	Час перевірки паролів (хв)	Час перебору (хв)	Загальний час (хв)	Кількість раундів	Час перевірки паролів (хв)	Час перебору (хв)	Загальний час (хв)
8	2.2	18.3	20.5	**12**	35.72	18.3	54.02
9	4.65	18.3	22.95	**13**	72.1	18.3	90.4
10	9.01	18.3	27.31	**14**	144.2	18.3	162.5
11	17.84	18.3	36.14	**15**	287.8	18.3	306.1

ISSN 2663 - 4023

Приклад застосування комбінованої атаки

Перш ніж розробляти комплексну стратегію захисту, потрібно розуміти, як зловмисник може використати перелічені вище атаки для отримання доступу до даних.

Нижче наведено схему поведінки зловмисника під час комбінованої атаки (рис. 8).

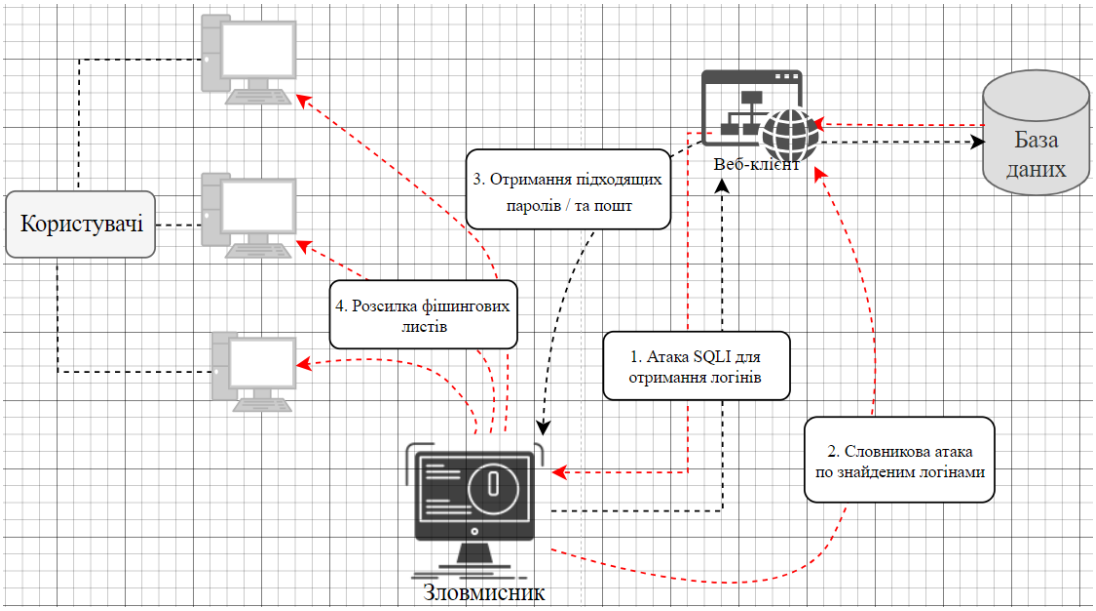


Рис. 8. Схема поведінки зловмисника при комбінованій атаці

На схемі видно, що все починається з SQLI-атаки на веб-клієнт. У цьому випадку головною задачею зловмисника є отримання інформації про користувачів, зокрема логінів (унікальних імен користувачів на платформі) та інших даних. У разі успіху зловмисник використовує словникову атаку для підбору паролів до знайдених логінів з метою успішної автентифікації. Після цього він пересилає собі всі зламані паролі та іншу отриману інформацію. Отримавши доступ до електронних поштових скриньок користувачів, зловмисник має можливість розсилати фішингові листів користувачам платформи. Крім того, якщо паролі, знайдені словниковою атакою, збігаються з тими, що використовуються для інших облікових записів, зловмисник зможе зламати і ці акаунти. У такому разі через скомпрометовані електронні скриньки також здійснюватиметься розсилка фішингових листів, а пов’язані з ними облікові записи з будуть під загрозою.

Проаналізувавши сценарій поведінки зловмисника, описаний у попередньому абзаці, стає зрозуміло, що комбіновані атаки можуть становити серйозну загрозу для сучасних комп’ютерних систем. З огляду на ці ризики важливим завданням для забезпечення інформаційної безпеки (ІБ) та мінімізації можливих інцидентів є розробка стратегії захисту від подібних атак.

Основними складовими цієї стратегії є захисні механізми, які слід впровадити у веб-застосунок для покращення його безпеки. У цій частині буде детально описано, які рішення були застосовані в роботі.

При розробці стратегії застосований підхід *Security Artichoke*. Це багатофакторний підхід, який являє собою певну сукупність захисних механізмів для мінімізації ризиків виникнення нових інцидентів ІБ. Кожен **«лист»** артишока є окремим захисним механізмом, незалежним від інших. Середина його є інформаційним активом, який треба захищати [15]. Нижче показано схему цього підходу (рис. 9).

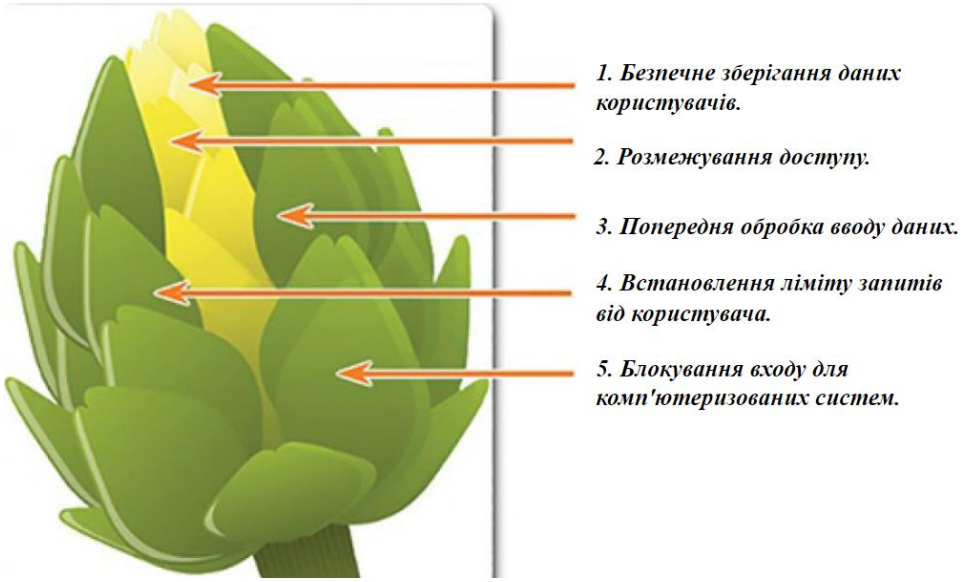


Рис. 9. Демонстрація підходу Security Artichoke

Особливу увагу слід приділити безпечному зберіганню даних, зокрема паролів. У стандарті ДСТУ ISO/IEC 27001:2015 [16], який встановлює вимоги до систем управління інформаційною безпекою (ІБ), питання захисту паролів та інших чутливих даних розглядаються в декількох розділах. Відповідно до стандарту, організації повинні впроваджувати відповідні заходи для захисту інформаційних активів, включаючи систему управління паролів. Це означає, що паролі мають зберігатися в прихованому вигляді, що реалізовується за допомогою сторонніх криптографічних функцій, таких як хешування.

У цій роботі для безпечного зберігання даних користувачів використовується бібліотека bcrypt для хешування паролів та JWT-токени для автентифікації й управління сесіями. Використовуюся два типи токенів:

Access-токени — мають короткий термін дії та застосовуються для доступу до захищених API-ендпоінтів.

Refresh-токени — діють довше та дозволяють оновити access-токен без необхідності повторного входу в систему.

Для ефективного розмежування доступу необхідно впровадити систему Role-Based Access Control (RBAC). Ця модель управління передбачає присвоєння кожному користувачеві певної ролі.

В цьому проєкті реалізовано наступну ієрархію ролей (рис. 10).

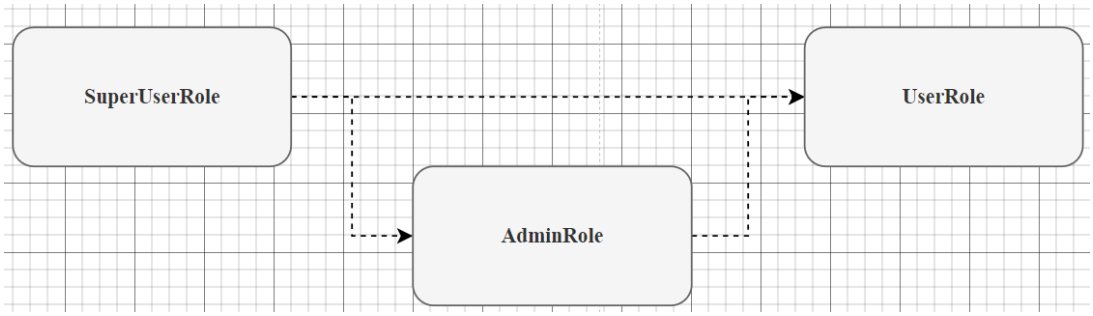


Рис. 10. Ієрархія ролей на проєкті

SuperUser — користувач із повними привілеями, який може виконувати будь-які дії в API проєкту. Зазвичай це керівник, що перевіряє якість виконаних завдань.

Admin — адміністратор платформи, який відповідає за наповнення контенту, може блокувати та розблоковувати користувачів із роллю User.

User — зареєстрований користувач із найменшими привілеями.

Також варто зазначити, що користувач із роллю SuperUser може не лише блокувати та розблоковувати Admin і User, а й змінювати ролі між ними. Реалізація цієї системи буде здійснена через модель *user* в базі даних, до якої буде додано два булеві поля *isAdmin* та *isSuperUser*. При виконанні певних функцій буде додано умовні перевірки *(if),* що визначити, чи має користувач відповідну роль.

Попередня обробка вводу даних (input validation) в цьому проєкті є критичним механізмом захисту інформаційних систем, що дозволить запобігти багатьом атакам, зокрема SQL-ін'єкціям (SQLI) та NoSQL-ін'єкціям (NoSQLI). У цьому проєкті валідація даних як на стороні клієнта, так і на сервері. Через спеціальні регулярні вирази буде проведена обробка на клієнті, яка буде включати обробку поля identifier імені користувача та електронної пошти, а також будуть встановлені вимоги до паролів. Під час введення даних також будуть виключені наступні спеціальні символи, які можуть бути використані в атаках. Важливим етапом стане параметризація запитів для мінімізації ризиків піддатися атаці SQLI чи NoSQLI. Параметризовний запит виглядає наступним чином:

```
const query = 'SELECT * FROM users WHERE username = ? OR email = ?';
connection.query(query, [identifier, identifier], async (err, rows) => {}
```

Запит використовує плейсхолдери «?» для параметрів identifier, що дозволяє передавати значення параметрів окремо в масиві під час виклику методу query().

Плейсхолдер — це спеціальний символ чи мітка, що використовується в SQL-запитах для позначення місця, де потрібно підставити значення параметра. Такий підхід унеможливлює SQL-ін'єкції (SQLI), оскільки всі вхідні дані коректно екрануються перед виконанням запиту до бази даних.

У рамках даної стратегії впроваджено два додаткові механізми безпеки: встановлення ліміту запитів та блокування входу для комп'ютеризованих систем, напрямленні проти атак на доступність, а також словникової атаки. Для захисту від атак на доступність використовується бібліотека rate-limiter-flexible. Цей механізм дозволяє обмежити кількість запитів, які може виконати користувач за певний період часу. Це допоможе запобігти перевантаженню сервера, захиститися від атаки brute-force та масового підбору паролів. Щоб заблокувати автоматизовані системи, використовується бібліотека та сервіс hCaptcha. Система випадковим чином генерує зображення і просить користувача виконати певне завдання (наприклад, вибрати всі зображення з авто). Це дозволяє визначити, чи є користувач людиною або ботом.

Після певної кількості невдалих спроб входу або підозрілих дій користувач перенаправляється на сторінку hCaptcha для перевірки.

Тестування впровадженої стратегії та висновки

Ця частина статті присвячена тестуванню реалізованих механізмів стратегії безпеки, описаних у попередніх розділах.

Після використання методу hash модуля bcrypt, паролі в базі даних стали зберігатися у хешованому вигляді (рис. 11), що є обов'язковою практикою при роботі з чутливими даними користувачів.

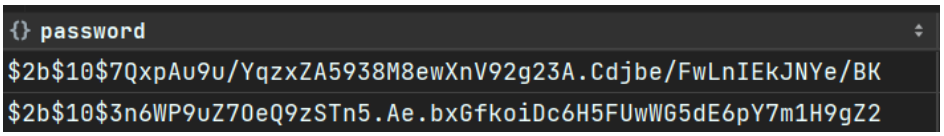


Рис. 11. Показ хешованих паролів користувачів

Даний механізм безпеки унеможливлює злам пароля користувача на основі його хешу. Згідно з документацією bcrypt [10], отримані хеші мають довжину 60 символів і містять сіль та інші параметри. Хешований пароль має наступну структуру:

```
$[алгоритм]$[коефіцієнт витрат]$[сіль][хеш]
```

Перші два символи ідентифікують алгоритм хешування (наприклад, «$2a$» або «$2b$» вказують на використання BCrypt).

Коефіцієнт витрат (cost factor) визначає кількість ітерацій, що розраховується як *2^n*.

Сіль (16 байтів, 128 біт) закодована у форматі base64 і займає 22 символи.

Хеш займає 24 байти (192 біти) та кодується у base64 до 31 символу.

Після впровадження хешування змінився і спосіб входу в обліковий запис:

при введенні пароля користувачем система хешує введене значення і порівнює його з хешем, що збережений у базі даних. Доступ до платформи надається тільки у разі збігу хешів.

За допомогою документації модуля JWT [17] було реалізовано генерацію токенів access та refresh, які повертаються у відповідь сервера (рис. 12).

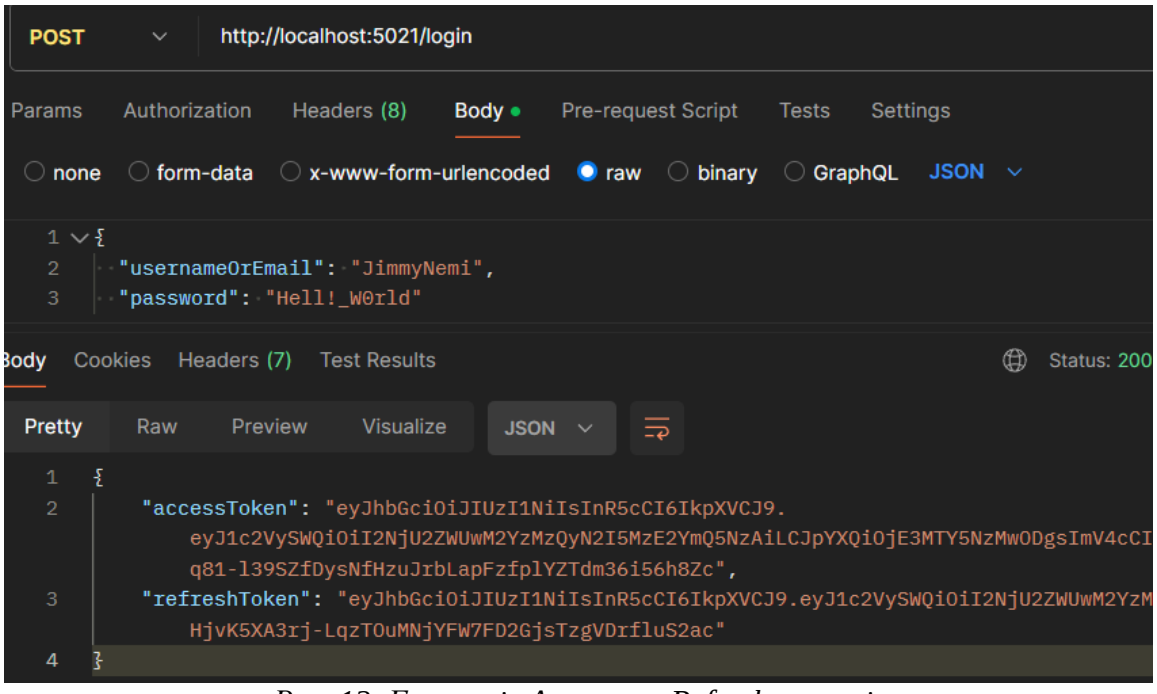


Рис. 12. Генерація Access та Refresh-токенів

Ці токени необхідні користувачу для взаємодії з функціоналом проєкту. А це значить що без них користувач не зможе жодним чином взаємодіяти з БД. Також варто зазначити що при написанні цієї статті використовувався функціонал для тестування API Postman. Цей функціонал дозволяє зручно тестувати запити до БД та сигналізувати про можливі помилки.

ISSN 2663 - 4023

Після впровадження управління доступом на основі ролей (RBAC) система перевіряє права доступу користувача перед виконанням запитів до API.

На прикладі нижче (рис. 13) показано спробу неавторизованого користувача отримати інформацію про користувача за id. Оскільки цей користувач не має токена доступу, система відмовляє у виконанні запиту.

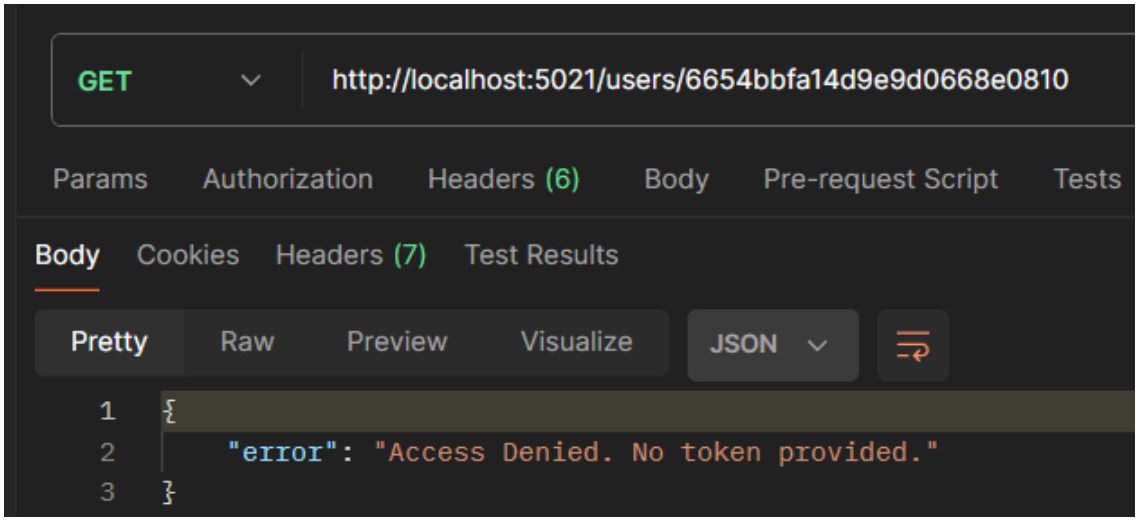


Рис. 13. Перевірка розмежованості доступу

Для уникнення загрози стосовно введення шкідливих NoSQLI чи SQLI проводиться попередня обробка полів для введення.

З веб-ресурсу regex101.com [18] було використано регулярні вирази для попередньої обробки полів пароля ^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@ $!%*?&]{8,}$, також ідентифікатора користувача: (^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$) — для електронної пошти та (^[a-zA-Z0-9_.+-]+$) — для імені користувача. Для застосування цих виразів найкраще підійшла бібліотека yup, яка створена для впровадження попередньої обробки. На основі її документації [19] створено об’єкт yup який має в собі метод matches (рис. 14), що дозволив вставити регулярні вирази та налаштувати під потрібний формат даних.

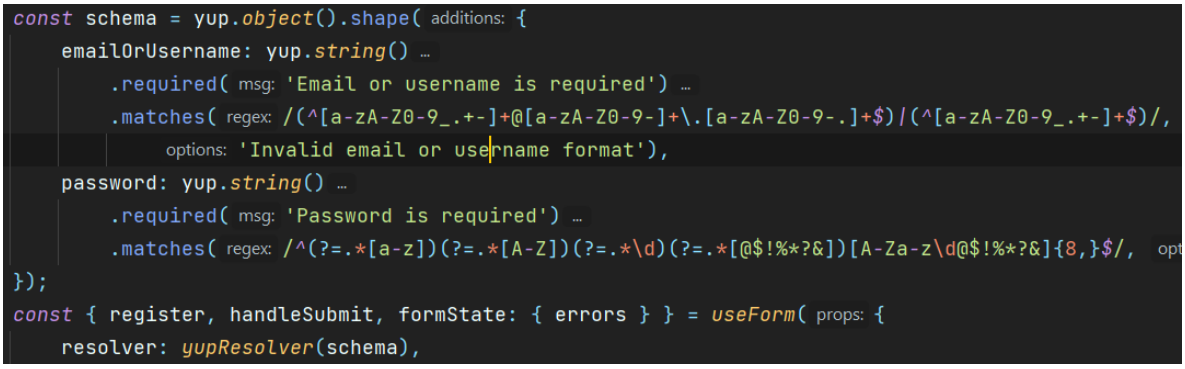


Рис. 14. Встановлення попередньої обробки полів користувача

На прикладі нижче (рис. 15) показано панель входу в систему, де було здійснено спробу атаки NoSQLI. Проте завдяки ефективній обробці введених даних, веб-застосунок не дозволяє передати шкідливий запит на сервер.

ISSN 2663 - 4023

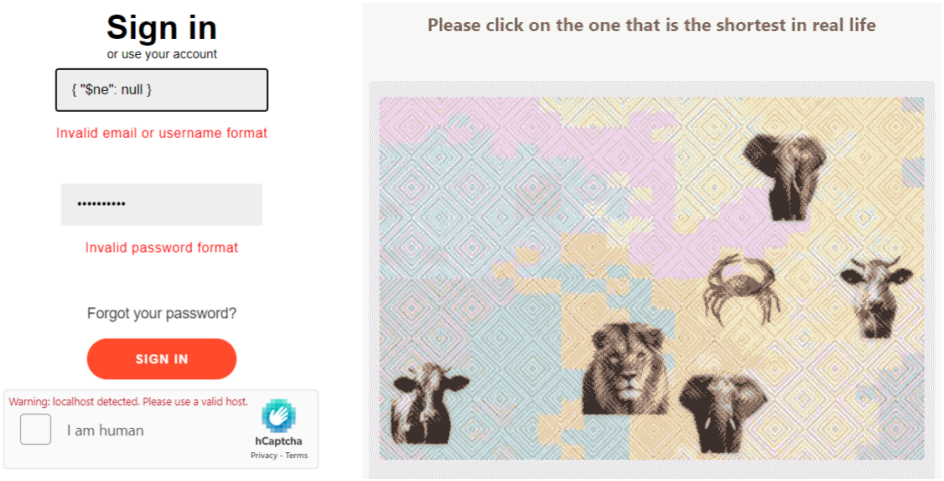


Рис. 15. Демонстрація оброблених полів панелі входу систему та задачі від hCaptcha

Також можна побачити перевірку hcaptcha. Перевірку зроблено таким чином, щоб уникнути реєстрації чи входженню до платформи комп'ютеризованих систем. В даному випадку застосовано безкоштовний план, де задача не є складною. Проте в платних версіях застосовані типи задач, які унеможливлюють проходження перевірки штучним інтелектом.

В кінці показано результат обмеження ліміту спроб для користувачів де було запущено словникову атаку проте після 5 спроб скрипт підхоплює помилку яка сигналізує про перевищення спроб. Таким чином словникова атака стає неефективною (рис. 16).

```
C:\Users\38063\IdeaProjects\dictionary_attack>node attack_script.js Charlie1 passwords.txt
Attempting login: Charlie1 with password: 1234568 Progress: 0%
Attempting login: Charlie1 with password: strong_password Progress: 0%
Attempting login: Charlie1 with password: abc123 Progress: 0%
Attempting login: Charlie1 with password: password Progress: 0%
Attempting login: Charlie1 with password: 123456 Progress: 0%
Attempting login: Charlie1 with password: computer Progress: 0%
Too Many Requests - waiting to retry...
```

Рис. 16. Результат невдалої словникової атаки

Реалізувавши серверні атаки та способи протидії їм, можна зробити порівняльний аналіз у вигляді таблиці, в якій вказано тип атаки та захід для протидії. Нижче складено порівняльну таблицю ефективності атак, де зазначено рівень ефективності атак до елементів стратегії захисту (табл. 2). Рівні ефективності поділено на: високий, середній та низький.

Атаки на основі ін'єкцій, такі як NoSQLI та SQLI, є неефективними, коли застосовано обробку введених даних, і менш ефективними, коли обмежено доступ до певних маршрутів API. Словникова атака є ефективною при розмежуванні доступу та при попередній обробці даних. Проте хешування паролів, встановлення ліміту запитів та блокування комп'ютеризованих систем значно ускладнить цю атаку.

Остання комбінована атака описана на початку, використовує комбінацію вище зазначених атак. Якщо в системі впроваджено всі заходи стратегії захисту, то така атака

стане неефективною. Однак якщо хоча б один із механізмів не впроваджено, система залишається вразливою. Наприклад, недостатня обробка введених даних призведе до SQLI або NoSQLI, а відсутність обмеження кількості запитів зробить систему вразливою до словникових атак.

Для забезпечення високого рівня інформаційної безпеки необхідно впровадити всі рекомендовані заходи.

Таблиця 2

Порівняльна таблиця ефективності серверних атак

Захист/Атака	**SQL-ін'єкція**	**NoSQL-ін'єкція**	**Словникова атака**	**Комбінована атака**
Безпечне зберігання паролів	Високий	Високий	Середній	Низький
Розмежування доступу	Середній	Середній	Високий	Низький
Обробка введених даних	Низький	Низький	Високий	Низький
Ліміт запитів	Високий	Високий	Низький	Низький
Блокування комп. систем	Високий	Високий	Низький	Середній

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У даній статті показано принцип дії серверних атак на реляційні та нереляційні бази даних, зокрема NoSQL-ін'єкцію SQL-ін'єкцію та словникову атаку. Результатом дослідження цих атак стала розробка стратегії безпеки, яка включала такі заходи, як: безпечне зберігання паролів за допомогою bcrypt та JWT, впровадження політики розмежування доступу (RBAC), встановлення ліміту запитів від користувача через rate-limiter-flexible, застосування попередньої обробки за допомогою регулярних виразі і модуля Yup та блокування автоматизованих систем через hCaptcha.

Після впровадження стратегії захисту було проведено порівняльний аналіз ефективності серверних атак відповідно до застосованих захисних механізмів. До цього аналізу також увійшла комбінована атака, описана у розділі про приклад застосування атак. Порівняння показало, що ефективність атак залежить від реалізації механізмів безпеки платформи, її вразливостей та наявності/відсутності необхідних заходів захисту. Використання всіх запропонованих заходів безпеки значно підвищує рівень інформаційної безпеки платформи, тоді як часткове застосування заходів може залишати можливість для комбінованих атак.

Що стосується перспектив подальших досліджень, то варто дослідити більшу кількість серверних атак на БД, а також розглянути різні варіації комбінованої атаки. Також варто дослідити методи адаптивного виявлення атак, зокрема використання алгоритмів машинного навчання для автоматичного аналізу підозрілих запитів та аномальної активності користувачів. Це дозволить підвищити рівень захисту та оперативно реагувати на нові загрози.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Connolly, T. M., & Begg, K. E. (2013). *Database Systems: The New Pearson International Edition: A Practical Approach to Design, Implementation, and Management.* Pearson Education, Limited.
2. Romanyuk, O. V., Denisyuk, A. V., Marushchak, A. V., & Shmalyuk, V. A. (2021). Comparative Analysis of SQL and NoSQL Databases. *In 12th International Scientific and Technical Conference "Information and Computer Technologies - 2021 (ICT - 2021)",* Zhytomyr Polytechnic University.
3. *NoSQL for Mere Mortals*®. (n.d.). O'Reilly Online Learning. https://www.oreilly.com/library/view/nosql-for-mere/9780134029894/
4. Subramanian, S., & Saravanan, S. (2024). Current trends in No SQL databases. *International Journal of Computer Trends and Technology, 72(9),* 126–130. https://doi.org/10.14445/22312803/ijctt-v72i9p119
5. Mongodb Injection Dataset: A Complete Collection of Mongodb – NoSQL Injection Attempts and Vulnerabilities. (n.d.). *Data Brief, 110289*. https://doi.org/10.1016/j.dib.2024.110289
6. Kumar, P. & Singh, R. (2024). *Security vulnerabilities in SQL databases: analysis and prevention mechanisms. Next-generation computer systems.* Elsevier.
7. monitorapp_admin. (2024). [2024.05] *Web attack trend report.* MONITORAPP. https://www.monitorapp.com/may-2024-web-attack-trend-report/
8. Oselsky, S. V. & Oselsky, S. (2019). *Methodology for protecting information confidentiality in mssql and mysql databases from sql attacks [Master's thesis]*. ELARTU – Institutional repository of Ivan Pulyuy TNTU. http://elartu.tntu.edu.ua/handle/lib/30595
9. O'Driscoll, A., & O'Driscoll, A. (2023). *25+ Password hacking statistics and trends (that may change your password habits).* Comparitech. https://www.comparitech.com/blog/information-security/password-statistics/
10. *What is SQL Injection? Tutorial and Examples. Web Security Academy.* (n.d.). https://portswigger.net/web-security/sql-injection
11. npm: mongoose. (n.d.). *Npm.* https://www.npmjs.com/package/mongoose
12. *NoSQL Injection.* (n.d.). Web Security Academy. https://portswigger.net/web-security/nosql-injection
13. International Standard ISO 27002. (2013). *Information Technology. Security Methods. Code of Practice for Information Security Management.* Kyiv: State Consumer Standards of Ukraine.
14. Daniel Missler. (n.d.). *SecLists/Passwords/darkweb2017-top10000.txt in master· danielmiessler/ SecLists. GitHub.* https://github.com/danielmiessler/SecLists/blob/master/Passwords/darkweb2017-top10000.txt
15. *CCNA Cyber Ops (Version 1.1) – Chapter 8: Protecting the Network.* (2019). ITexamAnswers.net. https://itexamanswers.net/ccna-cyber-ops-version-1-1-chapter-8-prot ecting -the-network.html
16. Information technologies. Protection methods. Information security management systems. Requirements (62498) (DSTU ISO/IEC 27001:2015) (n.d.). https://dnaop.com/html/62498/doc%D0%94%D0%A1%D0%A2%D0%A3_ISO_IEC_27001_2015
17. npm: jsonwebtoken. (n.d.). *Npm.* https://www.npmjs.com/package/jsonwebtoken
18. Dib, F. (n.d.*). regex101: build, test, and debug regex.* Regex101. https://regex101.com/
19. npm: yup. (n.d.). *Npm.* https://www.npmjs.com/package/yup

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (28), 2025
ISSN 2663 - 4023

Mykhailo Markevych
Student of the Department of Information Technology Security
Lviv Polytechnic National University, Lviv, Ukraine
ORCID ID 0009-0000-3228-3525
mykhailo.markevych.mkbbi.2024@lpnu.ua

Oleh Horiachyi
Assistant of the Department of Information Technology Security
Lviv Polytechnic National University, Lviv, Ukraine
ORCID ID 0000-0003-4948-458X
oleh.y.horiachyi@lpnu.ua

STUDY OF THE EFFECTIVENESS OF SERVER ATTACKS ON RELATIONAL AND NON-RELATIONAL DATABASES. CREATION OF A DEFENSE STRATEGY.

Abstract. The current state of information technology development is characterized by the widespread use of databases in various fields of activity, in particular, in business, medicine, science and public administration. The growth of data volumes and their value leads to an increase in the number of cyberattacks on databases. In this regard, the issue of ensuring database security is becoming particularly relevant. The article considers the effectiveness of server attacks on relational and non-relational databases. A detailed analysis of attack methods, such as SQL/NoSQL injections and dictionary attacks, is carried out, and their consequences for the security of information systems are assessed. The effectiveness of password brute-force cracking is analyzed depending on the parameters of hash functions and the number of hashing rounds of the bcrypt library. It is shown that with an increase in the number of hashing rounds, the computational stability of password selection increases, so each attempt takes more time to process. However, even with a limited dictionary and an effective brute-force method, the attack can be performed in a fairly short period of time if the hashing parameters are chosen incorrectly. Schematic drawings of attacks on the corresponding types of databases are presented. A comprehensive protection strategy is proposed to ensure the confidentiality, integrity and availability of information, which includes pre-processing of user data, hashing of passwords, setting limits on the number of requests, delimiting access and blocking of computerized actions. Methods for countering server attacks are described and implemented, in particular, functional libraries for the secure storage of user passwords are considered. In addition, a comparison of the effectiveness of different types of attacks in the context of existing protection methods is carried out. The results of the study demonstrated that the comprehensive implementation of the basic components of the protection strategy significantly increases the resistance of data to typical server attacks, especially in a scalable environment with a large number of entry points.

Keywords: database; information security; defense strategy; NoSQL injections; role-based access control; dictionary attack; combined attack; SQL injections.

REFERENCES (TRANSLATED AND TRANSLITERATED)

20. Connolly, T. M., & Begg, K. E. (2013). *Database Systems: The New Pearson International Edition: A Practical Approach to Design, Implementation, and Management.* Pearson Education, Limited.
21. Romanyuk, O. V., Denisyuk, A. V., Marushchak, A. V., & Shmalyuk, V. A. (2021). Comparative Analysis of SQL and NoSQL Databases. *In 12th International Scientific and Technical Conference "Information and Computer Technologies - 2021 (ICT - 2021)",* Zhytomyr Polytechnic University.
22. *NoSQL for Mere Mortals*®. (n.d.). O'Reilly Online Learning. https://www.oreilly.com/library/view/nosql-for-mere/9780134029894/
23. Subramanian, S., & Saravanan, S. (2024). Current trends in No SQL databases. *International Journal of Computer Trends and Technology, 72(9),* 126–130. https://doi.org/10.14445/22312803/ijctt-v72i9p119
24. Mongodb Injection Dataset: A Complete Collection of Mongodb – NoSQL Injection Attempts and Vulnerabilities. (n.d.). *Data Brief, 110289*. https://doi.org/10.1016/j.dib.2024.110289

25. Kumar, P. & Singh, R. (2024). *Security vulnerabilities in SQL databases: analysis and prevention mechanisms. Next-generation computer systems.* Elsevier.
26. monitorapp_admin. (2024). [2024.05] *Web attack trend report.* MONITORAPP. https://www.monitorapp.com/may-2024-web-attack-trend-report/
27. Oselsky, S. V. & Oselsky, S. (2019). *Methodology for protecting information confidentiality in mssql and mysql databases from sql attacks [Master's thesis]*. ELARTU – Institutional repository of Ivan Pulyuy TNTU. http://elartu.tntu.edu.ua/handle/lib/30595
28. O'Driscoll, A., & O'Driscoll, A. (2023). *25+ Password hacking statistics and trends (that may change your password habits).* Comparitech. https://www.comparitech.com/blog/information-security/password-statistics/
29. *What is SQL Injection? Tutorial and Examples. Web Security Academy.* (n.d.). https://portswigger.net/web-security/sql-injection
30. npm: mongoose. (n.d.). *Npm.* https://www.npmjs.com/package/mongoose
31. *NoSQL Injection.* (n.d.). Web Security Academy. https://portswigger.net/web-security/nosql-injection
32. International Standard ISO 27002. (2013). *Information Technology. Security Methods. Code of Practice for Information Security Management.* Kyiv: State Consumer Standards of Ukraine.
33. Daniel Missler. (n.d.). *SecLists/Passwords/darkweb2017-top10000.txt in master· danielmiessler/ SecLists. GitHub.* https://github.com/danielmiessler/SecLists/blob/master/Passwords/darkweb2017-top10000.txt
34. *CCNA Cyber Ops (Version 1.1) – Chapter 8: Protecting the Network.* (2019). ITexamAnswers.net. https://itexamanswers.net/ccna-cyber-ops-version-1-1-chapter-8-prot ecting -the-network.html
35. Information technologies. Protection methods. Information security management systems. Requirements (62498) (DSTU ISO/IEC 27001:2015) (n.d.). https://dnaop.com/html/62498/doc%D0%94%D0%A1%D0%A2%D0%A3_ISO_IEC_27001_2015
36. npm: jsonwebtoken. (n.d.). *Npm.* https://www.npmjs.com/package/jsonwebtoken
37. Dib, F. (n.d.*). regex101: build, test, and debug regex.* Regex101. https://regex101.com/
38. npm: yup. (n.d.). *Npm.* https://www.npmjs.com/package/yup