МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 2021 р.

На правах рукопису
УДК 003.26:004.056.55

ДИПЛОМНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: Програмний модуль захисту реляційних баз даних

Виконавець: Я.В. Шепеля

Науковий керівник: к.т.н., доцент Л.О. Терейковська

Нормоконтролер: к.т.н., доцент Л.О. Терейковська

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Бакалавр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ С.В. Казмірчук
«___» _____ 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи
студентки Шепелі Ярини Володимирівни

1. Тема: *Програмний модуль захисту реляційних баз даних*
затверджена наказом в.о. ректора від «18» січня 2021 р. № 44/ст.
2. Термін виконання з 29.12.2020 р. по 09.06.2021 р.
3. Вихідні дані: сучасні методи та технології захисту реляційних баз даних; відомі засоби захисту реляційних баз даних; за результатами проведеного аналізу визначити вимоги до модулю захисту реляційних баз даних; сформулювати етапи вирішення задачі розробки модулю захисту реляційних баз даних та реалізувати модуль програмно.
4. Зміст пояснювальної записки: аналіз сучасних технологій захисту реляційних баз даних; проектування програмного модулю захисту реляційних баз даних; розробка та дослідження програмного модулю.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	29.12.2020	*Виконано*
2.	Аналіз літературних джерел	20.01.2021	*Виконано*
3.	Обґрунтування вибору рішення	05.02.2021	*Виконано*
4.	Збір інформації	27.02.2021	*Виконано*
5.	Дослідження сучасних технологій захисту реляційних баз даних	15.03.2021	*Виконано*
6.	Дослідження механізмів реалізації SQL-ін’єкцій	28.03.2021	*Виконано*
7.	Розробка процедур протидії SQL-ін’єкціям	10.04.2021	*Виконано*
8.	Проектування архітектури модулю захисту	14.04.2021	*Виконано*
9.	Розробка програмного забезпечення та проведення експериментальних досліджень	03.05.2021	*Виконано*
10.	Перевірка на антиплагіат	20.05.2021	*Виконано*
11.	Оформлення і друк пояснювальної записки	26.05.2021	*Виконано*
12.	Оформлення презентації	04.06.2021	*Виконано*
13.	Отримання рецензій від рецензента	09.06.2021	*Виконано*

Дипломник _____ Я.В. Шепеля
(підпис, дата)

Дипломний керівник _____ Л.О. Терейковська
(підпис, дата)

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, загальним обсягом робота складає 54 сторінки, має 8 рисунків, 3 таблиці, 10 сторінок додатків. Список використаних джерел містить 27 найменувань і займає 3 сторінки.

Метою дипломної роботи є забезпечення захисту реляційних баз даних від SQL-ін’єкцій.

В роботі вирішено задачу побудови програмного модулю захисту реляційних баз даних, що за рахунок застосування розроблених процедур обробки клієнтських запитів, забезпечує можливість протидіє SQL-ін’єкціям.

В роботі проведено аналіз існуючих рішень в області захисту реляційних баз даних, здійснена формалізація механізмів реалізації SQL-ін’єкцій, розроблено процедури протидії найбільш розповсюдженим SQL-ін’єкціям, реалізоване проектування архітектури модулю захисту, розроблено відповідне програмне забезпечення та проведено тестування запропонованих рішень.

Розроблений модуль та програмне забезпечення відносяться до галузі інформаційної безпеки і можуть бути використані для підвищення рівня захищеності від кібератак, спрямованих на отримання несанкціонованого доступу до реляційних баз даних.

Основні напрямки розвитку даної роботи можуть бути пов’язані з вдосконаленням системи протидії SQL-ін’єкціям за рахунок застосування нейромережевих моделей та методів їх виявлення.

Ключові слова: база даних, модуль захисту, SQL-ін’єкція, SQL-запит.

ЗМІСТ

ВСТУП

Актуальність. В даний час практично жодна організація в своїй діяльності не обходиться без використання баз даних (БД). Оскільки в БД може зберігатися конфіденційна або навіть секретна інформація, що стосується фінансової документації і клієнтської інформації, актуальним є питання її захисту і безпеки. Під захистом БД розуміється спосіб попередження несанкціонованого доступу до інформації, що зберігається в таблицях такої БД. Довгий час захист БД асоціювався з захистом локальної мережі підприємства від зовнішніх атак хакерів, боротьбою з вірусами. Однак, одним з найбільш слабких місць при забезпеченні безпеки даних є велика кількість осіб, які отримують до них доступ на самих різних рівнях. Тому загрози виникають не тільки ззовні, але й зсередини, з боку легальних користувачів.

В більшості розповсюджених систем до основних засобів захисту інформації в БД відносять такі: парольний захист, встановлення прав доступу до об'єктів БД, захист полів і записів таблиць БД, шифрування даних і програм.

Парольний захист представляє простий і ефективний спосіб захисту БД від несанкціонованого доступу. Паролі встановлюються кінцевими користувачами або адміністраторами БД і зберігаються в системних файлах систем управління базами даних (СУБД) в зашифрованому вигляді.

Для контролю використання основних ресурсів СУБД в багатьох системах використовуються засоби встановлення прав доступу до об'єктів БД. Права доступу визначають можливі дії над об'єктами. Власник об'єкта, а також адміністратор БД мають всі права. Решта користувачів до різних об'єктів можуть мати різні рівні доступу. До даних, наявних в таблиці, можуть застосовуватись заходи захисту по відношенню до окремих полів і окремих записів. У реляційних СУБД окремі записи спеціально не захищаються. Стосовно захисту даних в полях таблиць можна виділити такі рівні прав доступу, як повна заборона доступу, тільки читання, дозвіл всіх операцій. Ще одним потужним засобом захисту даних від перегляду є їх шифрування.

Атаки на БД є одними з найнебезпечніших для підприємств і організацій. Згідно зі статистикою за останні роки кількість витоків даних в світі неухильно зростає, при цьому на 2015 рік понад тридцяти відсотків з них припадають на зовнішніх порушників і більше шістдесяти відсотків виконано з участю співробітників організації. Навіть якщо припустити, що в ряді випадків витік включали дані, до яких співробітник має легальний доступ, кожен третій випадок припадав на зовнішню атаку. Зловмисників цікавлять такі види інформації, як внутрішня операційна інформація, персональні дані співробітників, фінансова інформація, інформація про замовників/клієнтів, інтелектуальна власність, аналіз діяльності конкурентів, платіжна інформація. Ці відомості в підсумку зберігаються в корпоративних сховищах і БД різного об'єму.

Все це призводить до необхідності забезпечення захисту не тільки комунікацій, операційних систем та інших елементів інфраструктури, а й сховищ даних як ще одного бар'єра на шляху зловмисника. При цьому на сьогоднішній день роботи в галузі забезпечення безпеки БД спрямовані в основному на подолання існуючих та вже відомих вразливостей, реалізацію основних моделей доступу і розгляд питань, специфічних для конкретної СУБД. Однак у зв'язку з вибуховим зростанням успішних кібератак типу SQL-ін’єкцій все більшу важливість набувають питання розробки методів та засобів протидії таким кібератакам.

Метою дипломної роботи є забезпечення захисту реляційних баз даних від розповсюджених SQL-ін’єкцій.

Досягнення мети досліджень потребує розв’язання таких **задач**:

– аналіз існуючих рішень в області захисту реляційних баз даних;

– проектування модулю захисту реляційних баз даних від розповсюджених SQL-ін’єкцій;

– розробка та тестування програмного забезпечення модулю захисту реляційних баз даних від розповсюджених SQL-ін’єкцій.

Об'єкт дослідження: процеси захисту реляційних баз даних від SQL-

ін’єкцій.

Предмет дослідження: моделі, методи та засоби захисту реляційних баз даних від SQL-ін’єкцій.

Оцінка сучасного стану проблеми на основі вітчизняної та зарубіжної літератури. Питанням забезпечення інформаційної безпеки, в тому числі і з питанням захисту реляційних баз даних присвячені праці багатьох вітчизняних і зарубіжних вчених: Ю. Бортовчук, Е. Горбачевская, М. Егоров, О. Корченко, С. Кузнецов, А. Потапов, M. Murray, S. Rohilla, G. Wassermann. Однак проведений аналіз показує, що дана інформація дуже швидко застаріває, тому потрібне постійне оновлення актуальної інформації.

Аналіз цих публікацій вказує на те, що до недоліків системи захисту реляційних баз даних відносять недосконалість технології протидії SQL-ін’єкціям. Для подолання вказаного недоліку у відповідних системах захисту використовуються різноманітні синтаксичні аналізатори клієнтських запитів, що використовуються для формування звернень до реляційних баз даних. При цьому виникнення нових видів SQL-ін’єкцій потребує застосування в системах протидії додаткового аналізу даних, надісланих від клієнта.

Галузь застосування. Розроблений програмний модуль захисту реляційних баз даних від SQL-ін’єкцій може використовуватися у галузі захисту інформації, зокрема при розробці комплексної системи захисту веб-орієнтованих інформаційних систем.

Новизна. Отримала подальший розвиток технологія захисту реляційних баз даних від SQL-ін’єкцій.

Практична цінність полягає у тому, що запропонований програмний модуль може використовуватися в системах захисту веб-орієнтованих інформаційних систем від SQL-ін’єкцій. Окрім того, запропоновані практичні рекомендації можуть стати у нагоді експертам з інформаційної безпеки при прийнятті рішень щодо доцільності застосування різного роду засобів захисту баз даних.

Розділ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОЇ РОБОТИ

1.1. Проблема захисту реляційних баз даних

База даних може бути визначена як деяка сукупність логічно зв'язаних даних та описів цих даних. База даних – це єдине велике сховище даних, яке один раз визначається, а потім використовується одночасно багатьма користувачами з багатьох підрозділів. База даних зберігає не тільки самі дані, а також і описи цих даних. У сукупності описи даних називаються системним каталогом або словником даних, о окремий елемент словника – метаданим, тобто даним щодо даного. Саме наявність метаданих і забезпечує незалежність програм, які забезпечують роботу з базою даних, від самих даних.

Робота з базою даних здійснюється за допомогою системи керування або управління базами даних – СУБД. СУБД – це програмне забезпечення, за допомогою якого користувачі можуть визначати, створювати та підтримувати базу даних і здійснювати контрольований доступ до неї.

Визначення бази даних здійснюється за допомогою мови визначення даних, яка дає можливість користувачеві вказати тип даних та їх структуру, а також певні обмеження на інформацію, що зберігається в базі даних.

Введення, редагування, видалення та пошук даних, як правило, здійснюється за допомогою мови керування даними.

Для забезпечення контрольованого доступу до бази даних СУБД повинна включати:

- систему забезпечення безпеки, яка забороняє несанкціонований доступ до інформації;
- систему підтримки цілісності даних, яка забезпечує незаперечливий стан даних бази даних;
- систему керування паралельною роботою з базою даних;
- систему відновлення бази даних, яка дозволяє відновити стан бази даних до моменту, коли відбувалися негаразди з апаратним або програмним забезпеченням.

Середовище СУБД складається з п'яти компонентів:

1. Апаратне забезпечення.

2. Програмне забезпечення.

3. Дані.

4. Процедури.

5. Користувачі.

Апаратне забезпечення може бути представлено як одним комп'ютером, так і комп'ютерною мережею. Апаратне забезпечення залежить від організації, для якої розроблена база даних, та конкретної СУБД. Деякі СУБД розраховані на роботу з вибірковими комп'ютерами та операційними системами, інші розраховані на велике коло апаратного забезпечення та операційних систем.

Програмне забезпечення охоплює як програмне забезпечення самої СУБД, так і прикладне програмне забезпечення, операційну систему та програмне забезпечення комп'ютерної мережі, якщо СУБД використовується в мережі.

Дані – найважливіша компонента СУБД. В графічному вигляді дані часто відображають у вигляді моста, який з'єднує комп'ютер і людину. База даних містить як робочі дані, так і описи цих даних – метадані. Структура бази даних називається схемою бази даних.

До процедур відносяться інструкції та правила, які повинні враховуватися під час проектування та використання бази даних:

- реєстрація в СУБД;
- використання прикладного програмного забезпечення;
- запуск і зупинка СУБД;
- створення резервних копій;
- відновлення бази даних після можливих відмов системи тощо.

Щодо останньої п'ятої компоненти, то тут можна виділити чотири групи користувачів системи:

1. Адміністратор бази даних – відповідає за всю систему в цілому, за її безпеку, за апаратне та програмне забезпечення тощо.

2. Розробник бази даних – займається розробкою логічної структури бази даних та реалізацією логічної структури на фізичному рівні, вибором конкретних структур збереження даних та методів доступу до них, проектує засоби захисту даних.

3. Прикладний програміст – займається розробкою додатків до бази даних, які і забезпечують інформаційні потреби кінцевих користувачів.

4. Кінцевий користувач – клієнт бази даних, для яких вона створюється і підтримується.

Сучасна хвиля інформаційних технологій управління ґрунтується на використанні реляційних СУБД. Реляційні бази даних і технології клієнт-сервер є типовою комбінацією, що дозволяє сучасним компаніям успішно обробляти дані і залишатися конкурентоспроможними в своїх секторах ринку.

Реляційна база даних - це база даних, розділена на логічно цілісні сегменти, звані таблицями, і всередині бази даних ці таблиці пов'язані між собою. Реляційна база даних дозволяє розділити дані на логічні більш дрібні і більш керовані сегменти, що забезпечує оптимальне представлення даних і можливість організації декількох рівнів доступу до даних.

У будь-якій реляційній базі даних користувач сприймає базу даних як деяку сукупність таблиць. Кожна таблиця в реляційній моделі називається відношенням. Відношення має вигляд двомірної таблиці, рядки якої – окремі записи, а стовпчики з назвами – атрибути.

Сукупність значень, на якому визначаються атрибути, називається доменом. Іншими словами, домен визначає, які значення і якого типу можуть бути введені для атрибутів. Кількість атрибутів у відношенні отримала назву ступеня відношення. Кожний рядок відношення, тобто кожний рядок таблиці, має назву кортеж. Кортежі можуть бути записані в довільному порядку у відношенні, але від цього структура відношення не змінюється. Кількість кортежів у відношенні має назву кардинальність.

Таблиці в реляційної базі даних є пов'язаними, що дозволяє витягти тільки потрібні дані за допомогою одного запиту (хоча при цьому самі

необхідні дані можуть вилучатись з декількох таблиць). Внаслідок наявності у таблиць загальних ключів або, інакше, ключових полів, виявляється можливим об'єднати дані з декількох таблиць в одне результуюче безліч.

SQL - мова структурованих запитів - є стандартною мовою керування реляційними базами даних. Його прототип був розроблений фірмою IBM на основі ідей, викладених в статті д-ра Кодда (E. F. Codd) "Реляційна модель даних для великих банків даних загального користування". Трохи пізніше появи прототипу IBM, в 1979 році, на ринку з'явився перший продукт SQL під назвою ORACLE, який був випущений компанією Relational Software, Incorporated (згодом перейменованої в Oracle Corporation). Сьогодні ця компанія є одним з видатних лідерів в області реалізації технологій реляційних баз даних. Конкретні реалізації SQL різних виробників мають певні особливості. Сервери бази даних, подібно до будь-якого іншого продукту на ринку, виготовляються широким спектром виробників. У кожній реалізації SQL виробники пропонують різні удосконалення з метою спрощення роботи з виробленими ними серверами баз даних. Ці удосконалення або, розширення, являють собою команди і опції, запропоновані на додаток до стандартного набору команд SQL і доступні в рамках кожної конкретної реалізації. Сеанс SQL - це період взаємодії користувача з реляційною базою даних за допомогою використання команд SQL. Починається сеанс з моменту підключення користувача до бази даних. В рамках сеансу користувач має можливість вводити допустимі команди SQL для здійснення запитів, управління даними, створення нових структур бази даних.

Сеанс SQL починається в момент підключення користувача до бази даних. Для цього використовується команда CONNECT. За допомогою команди CONNECT можна або здійснювати підключення до бази даних, або міняти характер вже встановлених підключень. Наприклад, підключившись до бази даних під ім'ям USER1, ви можете потім використовувати команду CONNECT, щоб підключитися до тієї ж бази даних під ім'ям USER2. При цьому неявно припиняється сеанс SQL для користувача USER1.

При спробі підключитися до бази даних ви автоматично отримаєте запит на введення пароля, відповідного введеному вами імені користувача.

Сеанс SQL припиняється при відключенні користувача від бази даних. Для відключення користувача від бази даних використовується команда DISCONNECT. Після відключення від бази даних ви ще можете користуватися програмними засобами зв'язку з базою даних, але сама зв'язок з базою даних буде припинена. При використанні для розриву зв'язку оператора EXIT припиняється не тільки ваш сеанс SQL, але і закривається програма, за допомогою якої здійснювався доступ до бази даних.

В мові SQL виділяються категорії команд, що дозволяють реалізувати різноманітну функціональність щодо баз даних. Серед таких функцій - побудова об'єктів бази даних, управління об'єктами, поповнення таблиць бази даних новими даними, оновлення даних, вже наявних в таблицях, виконання запитів, управління доступом користувачів до бази даних, а також здійснення загального адміністрування бази даних. При цьому до вказаних категорій відносяться:

• DDL (Data Definition Language - мова визначення даних);

• DML (Data Manipulation Language - мова маніпуляцій даними);

• DQL (Data Query Language - мова запитів до даних);

• DCL (Data Control Language - мова керування даними);

• команди адміністрування даних;

• команди управління транзакціями.

В контексті баз даних термін безпека означає захист даних від несанкціонованого перегляду, зміни або знищення. Мова SQL дозволяє індивідуально захищати як цілі таблиці, так і окремі їх поля. Для цього є дві відносно незалежні можливості:

- Механізм подання, який використовується для приховування засекречених даних від користувачів, що не володіють правом доступу.

- Підсистема розподілу прав доступу, що дозволяє надати зазначеним легітимним користувачам певні привілеї на доступ до даних і дати їм

можливість вибірково і динамічно передавати частину виділених привілеїв іншим користувачам, скасовуючи згодом ці привілеї, якщо це буде потрібно.

Зазвичай при установці СУБД в неї вводиться якийсь ідентифікатор, який повинен далі розглядатися як ідентифікатор найбільш привілейованого користувача - системного адміністратора. Кожен, хто може увійти в систему з цим ідентифікатором (і може витримати випробування на достовірність), буде вважатися системним адміністратором до виходу з системи. Системний адміністратор може створювати бази даних і має всі привілеї на їх використання. Ці привілеї або їх частину можуть надаватися іншим користувачам (користувачам з іншими ідентифікаторами). У свою чергу, користувачі, які отримали привілеї від системного адміністратора, можуть передати їх (або їх частину) іншим користувачам, які можуть їх передати наступним і т.д.

Привілеї надаються за допомогою запиту GRANT (надати), загальний формат якого має вигляд:

GRANT привілеї ON об’єкт TO користувачі;

У даному запиті:

- "привілеї" - список, що складається з однієї або декількох привілеїв, розділених комами, або фраза ALL PRIVILEGES (всі привілеї);

- "об'єкт" - ім'я і, якщо треба, тип об'єкта (база даних, таблиця, подання, індекс і т.п.);

"користувачі" - список, що включає один або більше ідентифікаторів санкціонування, розділених комами, або спеціальне ключове слово PUBLIC (загальнодоступний).

До таблиць (уявлень) відносяться привілеї SELECT, DELETE, INSERT і UPDATE [(стовпці)], що дозволяють відповідно зчитувати (виконувати будь-які операції, в яких використовується SELECT), видаляти, додавати або змінювати рядки зазначеної таблиці (зміна можна обмежити конкретними стовпцями). Наприклад, запит

GRANT SELECT, UPDATE (Праця) ON Страви TO cook;

дозволяє користувачеві, який представився системі ідентифікатором cook, використовувати інформацію з таблиці «Страви», але змінювати в ній він може тільки значення стовпця «Праця».

Якщо користувач USER_1 надав будь-які привілеї іншому користувачеві USER_2, то він може згодом скасувати всі або деякі з цих привілеїв. Скасування здійснюється за допомогою запиту REVOKE (скасувати), загальний формат якого дуже схожий на формат пропозиції GRANT:

REVOKE привілеї ON об'єкт FROM користувачі;

Наприклад, можна відібрати у користувача cook право зміни значень стовпця «Праця» в таблиці «Страви». Для цього слід використати запит:

REVOKE UPDATE (Праця) ON Страви FROM cook;

Слід звернути увагу на те, що повноваження, надані групі public, є більш значущі, ніж повноваження, визначені раніше для деякого користувача або групи користувачів. Порядок видачі повноважень має значення. Повноваження, що видані останніми, мають переважаючий ефект. Таким чином, рекомендується спочатку видавати повноваження групі public, і далі по низхідній.

В табл.1.1 наведено список параметрів, що визначають дозволи в запитах GRANT та REVOKE.

Таблиця 1.1

Список параметрів, що визначають дозволи в запитах GRANT та REVOKE

Назва параметру	Опис дозволу
ALTER	Дозволяє використання ALTER TABLE
CREATE	Дозволяє використання CREATE TABLE
CREATE TEMPORARY TABLES	Дозволяє використання CREATE TEMPORARY TABLE
DELETE	Дозволяє використання DELETE
DROP	Дозволяє використання DROP TABLE
EXECUTE	Дозволяє користувачеві запускати процедури, що

	зберігаються

Таблиця 1.1 (продовження)

FILE	Дозволяє використання SELECT ... INTO OUTFILE и LOAD DATA INFILE
INDEX	Дозволяє використання CREATE INDEX and DROP INDEX
INSERT	Дозволяє використання INSERT
LOCK TABLES	Дозволяє використання LOCK TABLES в таблицях, для яких є привілей SELECT
PROCESS	Дозволяє використання SHOW FULL PROCESSLIST
RELOAD	Дозволяє використання FLUSH
REPLICATION CLIENT	Дозволяє користувачеві право на запит про місцезнаходження головного та підпорядкованих серверів
SELECT	Дозволяє використання SELECT
SHUTDOWN	Дозволяє використання mysqladmin shutdown
SUPER	Дозволяє встановити одне з'єднання, навіть за умови не перевищення встановленого значення max_connections. Також дозволяє запуск команд CHANGE MASTER, KILL thread, mysqladmin debug, PURGE MASTER LOGS та SET GLOBAL
UPDATE	Дозволяє використання UPDATE

В табл.1.2 наведено список параметрів, що визначають додаткові опції привілеїв в запитах GRANT та REVOKE.

Зазначимо, що визначені в стандарті мови SQL запити GRANT та REVOKE в основному використовуються для розподілу прав доступу до реляційних баз даних. Крім того, безпека баз даних багато в чому залежить від безпеки системного каталогу.

Системний каталог - це набір таблиць, в яких міститься інформація, необхідна для правильного функціонування СУБД. У різних СУБД, що

підтримують SQL, існує від десятка до декількох десятків системних таблиць, структура яких нічим не відрізняється від вже знайомої нам структури користувальницьких таблиць.

Таблиця 1.2

Параметри запитів GRANT та REVOKE, що визначають додаткові опції привілеїв

Назва параметру	Опис дозволу
ALL [PRIVILEGES]	Задає прості привілеї, крім WITH GRANT OPTION
REFERENCES	Зарезервовано для подальшого використання
REPLICATION SLAVE	Використовується для підпорядкованих серверів при реплікації (для читання інформації із бінарних журналів головного сервера)
SHOW DATABASES	Виводить перелік доступних баз даних
USAGE	Синонім для ``без привілеїв''
GRANT OPTION	Синонім для WITH GRANT OPTION

Так, в кожному рядку системної таблиці SYSTABLES зберігається опис однієї з таблиць призначених для користувача або системної баз даних. Для кожної з них вказується ім'я таблиці, ім'я користувача, який створив цю таблицю, число стовпців в ній і ряд інших елементів інформації. У таблиці SYSCOLUMNS міститься рядок для кожного стовпця кожної таблиці, в якій зазначено ім'я стовпця, ім'я таблиці, частиною якої є даний стовпець, тип даних для цього стовпця і багато іншої інформації про стовпці. За допомогою запиту SELECT користувач може отримати інформацію з будь-якої системної таблиці. Наприклад, він може дати запит на отримання імен таблиць, числа їх стовпців і рядків, власника і короткого опису (якщо таке вводилося в базу даних):

SELECT Tab_name, N_col, N_row, Tab_owner, Comments

FROM SYSTABLES;

Глобальні привілеї можна задати, скориставшись синтаксисом ON *.*, а привілеї бази даних - за допомогою синтаксису ON db_name.*.

Якщо вказати ON * при відкритій поточній БД, то привілеї будуть задані для цієї БД. При цьому в операторах GRANT та REVOKE шаблонні символи '_' і '%' в визначенні імені БД не допускаються. Це означає, що при необхідності використання в імені БД, скажімо, символу '_', необхідно вказати його як '\ _'.

1.2. Підходи до захисту реляційних баз даних

В сучасних СУБД підтримується один з двох найбільш загальних підходів до питання забезпечення безпеки даних: вибірковий підхід і обов'язковий підхід. В обох підходах одиницею даних або "об'єктом даних", для яких повинна бути створена система безпеки, може бути як вся база даних цілком, так і будь-який об'єкт всередині бази даних.

Ці два підходи відрізняються наступними властивостями:

• У разі виборчого управління деякий користувач володіє різними правами (привілеями чи повноваженнями) при роботі з цими об'єктами. Різні користувачі можуть мати різні права доступу до одного і того ж об'єкту. Виборчі права характеризуються значною гнучкістю.

• У разі невибіркового управління, навпаки, кожному об'єкту даних присвоюється певний класифікаційний рівень, а кожен користувач володіє деяким рівнем допуску. При такому підході доступом до певного об'єкту даних мають тільки користувачі з відповідним рівнем допуску.

Для реалізації виборчого принципу передбачені наступні механізми. У базу даних вводиться новий тип об'єктів БД - користувачі. Кожному користувачеві в БД присвоюється унікальний ідентифікатор. Для додаткового захисту кожному користувачеві крім унікального ідентифікатора присвоюється унікальний пароль. Причому якщо ідентифікатори користувачів в системі доступні системному адміністратору, то паролі користувачів зберігаються найчастіше в спеціальному зашифрованому вигляді і відомі тільки самим

користувачам.

Користувачі можуть бути об'єднані в спеціальні групи користувачів. Один користувач може входити в кілька груп. У стандарті вводиться поняття групи PUBLIC, для якої повинен бути визначений мінімальний стандартний набір прав. За умовчанням передбачається, що кожен новий користувач, якщо спеціально не вказано інше, відноситься до групи PUBLIC. Привілеї або повноваження користувачів або груп - це набір дій (операцій), які вони можуть виконувати над об'єктами БД.

В останніх версіях ряду комерційних СУБД з'явилося поняття "ролі". Роль - це пойменований набір повноважень. Існує ряд стандартних ролей, які визначені в момент установки сервера баз даних. Також передбачена можливість створювати нові ролі, групуючи в них довільні повноваження. Введення ролей дозволяє спростити управління привілеями користувачів, структурувати цей процес. Крім того, введення ролей не пов'язане з конкретними користувачами, тому ролі можуть бути визначені і сконфігуровані до того, як визначені користувачі системи. Користувачу може бути призначена одна або кілька ролей. Об'єктами БД, які підлягають захисту, є всі об'єкти, що зберігаються в БД: таблиці, представлення, збережені процедури і тригери. Для кожного типу об'єктів є свої дії, тому для кожного типу об'єктів можуть бути визначені різні права доступу.

На самому елементарному рівні концепції забезпечення безпеки БД достатньо прості. Необхідно забезпечити два фундаментальних принципи: перевірку повноважень і перевірку автентичності (аутентифікацію). Перевірка повноважень заснована на тому, що кожному користувачеві або процесу інформаційної системи відповідає набір дій, які він може виконувати по відношенню до певних об'єктів. Перевірка справжності означає достовірне підтвердження того, що користувач або процес, який намагається виконати санкціоноване дію, дійсно той, за кого він себе видає. Система призначення повноважень має в деякому роді ієрархічний характер. Найвищими правами і повноваженнями володіє системний адміністратор або адміністратор сервера

БД. Традиційно тільки цей тип користувачів може створювати інших користувачів і наділяти їх певними повноваженнями. СУБД в своїх системних каталогах зберігає як опис самих користувачів, так і опис їх привілеїв по відношенню до всіх об'єктів. Далі схема надання повноважень будується за наступним принципом. Кожен об'єкт в БД має власника - користувача, який створив цей об'єкт. Власник об'єкта має всі права-повноваженнями на даний об'єкт, в тому числі він має право надавати іншим користувачам повноваження по роботі з даним об'єктом або забирати у користувачів раніше надані повноваження.

У ряді СУБД вводиться найвищий рівень ієрархії користувачів - адміністратор БД. У цих СУБД один сервер може управляти безліччю БД (наприклад, MS SQL Server, Sybase). У СУБД Oracle застосовується однобазова архітектура, тому там вводиться поняття підсхеми - частини загальної схеми БД і вводиться користувач, який має доступ до підсхеми. У стандарті SQL не визначена команда створення користувача, але практично у всіх комерційних СУБД створити користувача можна не тільки в інтерактивному режимі, але і програмно з використанням спеціальних процедур, що зберігаються. Однак для виконання цієї операції користувач повинен мати право на запуск відповідної системної процедури.

Користувачів можна додавати двома різними способами - за допомогою команди GRANT або безпосередньо в таблиці призначення привілеїв СУБД. Переважно використовувати команду GRANT - цей спосіб простіший і дає менше помилок.

Після установки СУБД початкові привілеї доступу, як правило, задаються за допомогою спеціальних скриптів. Так для СУБД MySQL Скрипт mysql_install_db запускає сервер mysqld, а потім ініціалізує таблиці надання привілеїв з наступним набором привілеїв:

- В якості адміністратора СУБД створюється користувач root який може робити все, що завгодно. З'єднання повинні встановлюватися з локального комп'ютера. Спочатку пароль root порожній, тому будь-хто може приєднатися в

якості root без пароля і отримати всі привілеї.

- Створюється анонімний користувач, який може виконувати будь-які операції над БД з іменами test або починаються з test_. З'єднання повинні встановлюватися з локального комп'ютера. Це означає, що будь-який локальний користувач може підключитися без паролю і буде сприйнятий сервером як анонімний користувач.

- Решта привілеї заборонені. Наприклад, звичайний користувач не може використовувати команду mysqladmin shutdown.

При запуску сервера СуБД MySQL всі привілеї зчитуються в пам'ять. Привілеї бази даних, таблиці та стовпця вступають в силу негайно, а привілеї рівня користувача - при наступному під'єднанні користувача. Зміни в таблицях призначення привілеїв, які здійснюються за допомогою команд GRANT і REVOKE, обробляються сервером негайно. Якщо змінювати таблиці призначення привілеїв вручну (використовуючи команди INSERT, UPDATE і т.д.), необхідно запустити оператор FLUSH PRIVILEGES або mysqladmin flush-privileges, щоб вказати серверу на необхідність перезавантаження таблиць призначення привілеїв.

Відразу після установки СУБД слід встановити новий пароль для адміністратора. Для цього передбачена функції PASSWORD (). може не мати паролю, насамперед необхідно задати пароль. Для адміністратора СУБД MySQL це можна зробити наступним чином:

shell> mysql -u root mysql

mysql> SET PASSWORD FOR root @ localhost = PASSWORD ('new_password');

Більшість сучасних СУБД забезпечують захищений зв’язок між сервером та клієнтом. Захист реалізується шляхом шифрування, розшифровування даних. Шифрування - це метод, що дозволяє зробити прочитання будь-яких даних неможливим. Фактично при сучасному стані справ для алгоритмів шифрування потрібне використання додаткових елементів безпеки. Вони повинні забезпечувати протидію багатьох видів відомих на даний момент атак, таких як

зміна порядку зашифрованих повідомлень або повторення даних. Тому в сучасних СУБД використовуються апробовані протоколи шифрування.

Наприклад, одна із найбільш поширених СУБД MySQL підтримує шифровані SSL-з'єднання. При цьому за замовчуванням в MySQL використовуються незашифровані з'єднання між клієнтом і сервером. Це означає, що переглядати всі дані, що передаються між клієнтом і сервером, може хто завгодно. На практиці можна навіть змінювати дані під час передачі їх від клієнта до сервера і навпаки. Крім того, іноді виникає необхідність передати дійсно секретні дані через загальнодоступну мережу - в таких випадках використання незашифрованих з'єднань просто неприйнятно.

У протоколі SSL (Secure Sockets Layer) використовуються різні алгоритми шифрування, що забезпечують безпеку для даних, переданих через загальнодоступні мережі. Цей протокол містить засоби, що дозволяють виявляти будь-які зміни, втрати і повтори даних.

Протокол дозволяє передавати зашифровану інформацію по незасекреченим каналах, забезпечуючи надійний обмін між двома додатками, що працюють віддалено. Протокол складається з декількох шарів. Перший шар - це транспортний протокол TCP, що забезпечує формування пакета і безпосередню передачу даних по мережі. Другий шар - це захисний SSL Record Protocol. При захищеної передачі даних ці два шари є обов'язковими, формуючи якесь ядро SSL, на яке в подальшому накладаються інші шари. Наприклад, це може бути SSL Handshake Protocol, що дозволяє встановлювати відповідність між ключами і алгоритмами шифрування. Для посилення захисту переданої інформації на SSL можуть накладатися інші шари.

Для шифрування даних використовуються криптографічні ключі різного ступеня складності – 40-, 56- і 128-бітові. Показник кількості біт відображає стійкість застосовуваного шифру, його надійність. Найменш надійними є 40-бітові ключі, так як методом прямого перебору їх можна розшифрувати протягом 24 годин. У стандартному браузері Internet Explorer за замовчуванням використовуються 40- і 56-бітові ключі. Якщо ж інформація, що передається

користувачами, занадто важлива, то використовується 128-бітове шифрування. 128-бітові криптографічні ключі передбачені тільки для версій, що поставляються в США і Канаду. Для того, щоб забезпечити надійний захист, вам слід завантажити додатковий пакет безпеки - security pack.

Для передачі даних за допомогою SSL на сервері необхідна наявність SSL-сертифікату, який містить відомості про власника ключа, центрі сертифікації, дані про відкритий ключ (призначення, сфера дії і т. д.). Сервер може вимагати від користувача надання клієнтського сертифіката, якщо це передбачає використовуваний спосіб авторизації користувача.

При використанні сертифіката SSL сервер і клієнт обмінюються вітальними повідомленнями ініціалізації, що містять відомості про версії протоколу, ідентифікатор сесії, способі шифрування і стиснення. Далі сервер відсилає клієнтові сертифікат або ключове повідомлення, при необхідності вимагає клієнтський сертифікат. Після декількох операцій відбувається остаточне уточнення алгоритму і ключів, відправка сервером фінального повідомлення і, нарешті, обмін секретними даними. Такий процес ідентифікації може займати чимало часу, тому при повторному з'єднанні зазвичай використовується ідентифікатор сесії попереднього з'єднання.

Зазначимо, що в сучасних СУБД також можуть використовуватись більш сучасні протоколи шифрування даних, наприклад TLS. TLS використовує криптографію з відкритим ключем, яка дозволяє вузлам встановити загальний секретний ключ шифрування без будь-яких попередніх знань один про одного. Також в рамках процедури TLS Handshake, що забезпечує узгодженість параметрів з’єднання забезпечується можливість встановлення справжності особи клієнта і сервера. Нарешті, TLS забезпечує відправку кожного повідомлення з кодом MAC (Message Authentication Code), алгоритм створення якого - одностороння криптографічний функція хешування (фактично - контрольна сума), ключі якої відомі обом учасникам зв'язку. Всякий раз при відправленні повідомлення, генерується його MAC-значення, яке може згенерувати і приймач, це забезпечує цілісність інформації та захист від її

підміни.

Ще одним засобом забезпечення безпеки СУБД є аудит подій, що відбуваються в процесі її експлуатації. Дані про вказані події зберігаються в так званих журналах подій. Для прикладу в табл. 1.3 наведені характеристики типових журналів, що ведуться системою аудиту сучасних СУБД.

Таблиця 1.3

Характеристики журналів СУБД

Тип журналу	Призначення журналу
Журнал помилок	Збереження помилок запуску, роботи або завершення роботи.
Загальний журнал запитів	Збереження інформації про встановлені з'єднаннях і виконаних запитах.
Журнал оновлень	Збереження інформації про команди, зо вносять будь-які зміни в БД.
Журнал повільних запитів	Збереження інформації про всі запити, на виконання яких пішло більше часу, ніж це встановлено за замовчуванням

Крім того більшість сучасних СУБД дозволяє створювати резервні копії БД. Так в СУБД MySQL для створення резервних копій та структур БД використовуються утиліти mysqldump та mysqlhotcopy.

Утиліта mysqldump надає досить зручний спосіб дампування вмісту і структур таблиць. Зауважимо, що mysqldump це зручний засіб для копіювання структур і вмісту таблиць, які потім необхідно використовувати для завантаження в базу даних іншого SQL-сервера, причому не обов'язково MySQL.

Утиліта mysqldump може застосовуватися для резервного копіювання всіх баз даних, кількох баз даних, однієї бази даних, або навіть окремих таблиць конкретної бази даних. У наступних прикладах цього розділу наведено синтаксис команд, який використовується в кожному із цих випадків:

- Для створення резервної копії однієї бази даних:

%> Mysqldump [options] db_name

- Для резервування кілька таблиць в межах бази даних:

%> Mysqldump [options] db_name table1 table2. . . tableN

- Для резервування кількох баз даних:

%> Mysqldump [options] --databases [options] db_name1 . . . db_nameN

- Для резервування всіх баз даних:

%> Mysqldump [options] --all-databases [options]

Базуючись на [3, 11, 14], визначено, що зовнішніми дестабілізуючими факторами, що створюють загрозу безпеці функціонування СУБД, є:

- навмисні, деструктивні дії осіб з метою спотворення, знищення або викрадення програм, даних і документів системи, причиною яких є порушення інформаційної безпеки об'єкта, що захищається;

- спотворення в каналах передачі інформації, що надходить від зовнішніх джерел;

- збої і відмови в апаратурі обчислювальних засобів;

- віруси і інші деструктивні програмні елементи, поширювані з використанням систем телекомунікацій, що забезпечують зв'язок із зовнішнім середовищем або внутрішні комунікації розподіленої СУБД;

- зміни складу і конфігурації комплексу взаємодіє апаратури системи за межами, перевіреними при тестуванні або сертифікації системи.

Серед внутрішніх загроз можна виділити наступні атаки:

- атаки з боку авторизованих користувачів, спрямовані на підвищення привілеїв в СУБД;

- ненавмисні помилки співробітників, які з якої-небудь причини порушують встановлену політику безпеки або застосовують невірні методи безпеки;

- цілеспрямована зміна або спотворення даних, що зберігаються;

- загрози, що виникають через помилки в програмному та апаратному забезпеченні і невірну конфігурацію системи.

При цьому одними із найбільш небезпечних є атаки, спрямовані на розкриття конфіденційної інформації, що реалізуються за допомогою механізму SQL-ін’єкцій. На протязі останніх 5 років SQL-ін’єкції входять до 10 найбільш небезпечних атак та становлять біля 30% всіх успішних атак на СУБД.

Враховуючи наведену статистику та базуючись на результатах проведеного аналізу можна зробити висновок про те, що з точки зору протидії SQL-ін’єкціям, засоби захисту сучасних реляційних СУБД потребують доопрацювання.

1.3. Формулювання задач дипломної роботи

В результаті проведеного дослідження можна сформулювати такі вимоги до програмного модулю захисту реляційних баз даних:

– забезпечення ефективної протидії SQL-ін’єкціям;

– відповідність сучасним вимогам до надійності, швидкості та ресурсоємності;

– програмний додаток повинен бути орієнтований на використання в комп’ютерних системах, що функціонують під управлінням сучасних операційних систем.

Крім того, проведений аналіз задачі розробки програмного модулю захисту реляційних баз даних дозволяє виділити такі етапи її вирішення:

1. Побудова архітектури модулю захисту реляційних баз даних.
2. Дослідження механізму реалізації SQL-ін’єкцій для визначення прийомів протидії.
3. Розробка процедури захисту від SQL-ін’єкцій.
4. Розробка програмного забезпечення
5. Проведення експериментальних досліджень для підтвердження отриманих результатів.

Розділ 2. ПРОЕКТУВАННЯ МОДУЛЮ ЗАХИСТУ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

2.1. Механізми реалізації SQL-ін’єкцій

SQL ін'єкція - це атака, яка задіює динамічні оператори SQL, виносячи в коментарі певні частини інструкцій або додаючи умови, котрі завжди будуть істинним. Вона націлена на дірки в архітектурі веб-орієнтованих реляційних СУБД і використовує оператори SQL для виконання шкідливого SQL-коду. В більшості випадків SQL-ін'єкція - це атака, спрямована на веб-додаток, в ході якої конструюється SQL-вираз з призначеного для користувача введення шляхом простої конкантенації. Наприклад,

*$query = "SELECT * FROM users WHERE id =". $ _ REQUEST ["id"]*

У разі успіху атакуючий може змінити логіку виконання SQL-запиту так, як це йому потрібно. Найчастіше він виконує простий fingerprinting СУБД, а також витягує таблиці з найбільш "цікавими" іменами (наприклад "users"). Після цього, в залежності від привілеїв, з якими запущено вразливий додаток, він може звернутися до захищених частин серверної частини веб-додатку (наприклад, прочитати файли на стороні хоста або виконати довільні команди).

Типи атак, які можуть бути виконані з використанням SQL-ін'єкції, розрізняються за типом механізмів бази даних, що підлягають враженню. Як вже було зазначено, атака націлюється на динамічні оператори SQL. Динамічний оператор - це оператор, який створюється під час виконання на основі параметрів з веб-форми або рядка запиту URI. Розглянемо основні класи SQL-ін'єкцій:

- UNION query SQL injection. Оператор SQL UNION об'єднує результати двох або більше запитів і створює набір результатів, який включає витягнуті рядки з беруть участь запитів в UNION.

Основні правила об'єднання двох або більше запитів з використанням UNION:

1) Кількість стовпців і порядок стовпців всіх запитів повинні бути

однаковими.

2) Типи даних стовпців для включення таблиці в кожному запиті повинні бути однаковими або сумісними.

3) Зазвичай повертаються імена стовпців беруться з першого запиту. Класичний варіант впровадження SQL-коду, коли у вразливий параметр передається вираз, що починається з "UNION ALL SELECT". За замовчуванням UNION поводиться як UNION [DISTINCT], тобто усуває дублікати рядків; однак використання ключового слова ALL з UNION повертає всі рядки, включаючи дублікати. Ця техніка працює, коли веб-додатки безпосередньо повертають результат виведення команди SELECT на сторінку: з використанням циклу for або схожим способом, так що кожен запис отриманої з БД вибірки послідовно виводиться на сторінку. Зловмисник може також експлуатувати ситуацію, коли повертається тільки перший запис з вибірки (Partial UNION query SQL injection).

Припустимо, що зловмисник вводить «UNION SELECT * FROM emp_details» - в поле «Ідентифікатор користувача» і «abcd» в поле «Пароль», в якості імені користувача та пароля, який генерує наступний запит:

*SELECT * FROM user_details WHERE userid ='' UNION SELECT * FROM EMP_DETAILS -- ' and password = 'abcd'*

Дві риски (-) коментують решту запиту, тобто 'і пароль =' abcd '. Отже, запит стає об'єднанням двох запитів SELECT. Перший запит SELECT повертає нульовий набір, оскільки в таблиці user_details немає відповідного запису. Другий запит повертає всі дані з таблиці emp_details. Результатом виконання такого запиту є виведення персональних даних всіх користувачів БД.

- Error-based SQL injection. У разі цієї атаки зловмисник замінює або додає в вразливий параметр синтаксично неправильний вираз, після чого парсить HTTP-відповідь (заголовки і тіло) в пошуку помилок DBMS, в яких містилася б заздалегідь відома впроваджена послідовність символів і десь "поряд" висновок на цікавий для зловмисника підзапит. Ця техніка працює тільки тоді, коли веб-додаток з якихось причин (найчастіше з метою

налагодження) розкриває помилки DBMS.

- Stacked queries SQL injection. Зловмисник перевіряє, чи підтримує веб-додаток послідовні запити, і, якщо вони виконуються, додає у вразливий параметр HTTP-запиту крапку з комою (;) і слідом впроваджуваний SQL-запит. Цей прийом в основному використовується для впровадження SQL-команд, відмінних від SELECT, наприклад для маніпуляції даними (за допомогою INSERT або DELETE). Зазначається, що техніка потенційно може привести до можливості читання/запису з файлової системи сервера, а також виконання команд в ОС. Дієвість запиту значно залежить від використаної СУБД, а також призначених для користувача привілеїв.

- Boolean-based blind SQL injection. Реалізація так званої сліпий ін'єкції. Дані з БД в "чистому" вигляді вразливим веб-додатком ніде не повертаються. Прийом також називається дедуктивним. Зловмисник додає в вразливий параметр HTTP-запиту синтаксично правильно складений вираз, що містить підзапит SELECT (або будь-яку іншу команду для отримання вибірки з БД). Для кожної отриманої HTTP-відповіді виконується порівняння headers/body сторінки з відповіддю на початковий запит.

Таким чином, зловмисна утиліта може символ за символом визначити відповідь впровадженого SQL-виразу. В якості альтернативи користувач може надати рядок або регулярний вираз для визначення "true"-сторінок (звідси і назва атаки). Алгоритм бінарного пошуку, реалізований в деяких зловмисних утилітах для виконання цієї техніки, здатний максимум за сім HTTP-запитів витягти кожен символ виведення. У тому випадку, коли вивід складається не тільки зі звичайних символів, зловмисний сканер здатен підлаштовувати алгоритм для роботи з більш широким діапазоном символів (наприклад для unicode'a).

- Time-based blind SQL injection. Повністю сліпа ін'єкція. Точно так само як і в попередньому випадку, зловмисний сканер "грає" з вразливим параметром. Але в цьому випадку додає підзапит, який призводить до паузи в роботі DBMS на певну кількість секунд (наприклад, за допомогою команд

SLEEP() або BENCHMARK()). Використовуючи цю особливість, зловмисник може посимвольно витягти дані з БД, порівнюючи час відповіді на оригінальний запит і на запит з впровадженим кодом.

Тут також використовується алгоритм двійкового пошуку. Крім того, застосовується спеціальний метод для верифікації даних, щоб зменшити ймовірність неправильного вилучення символу через нестабільне з'єднання.

- Stored procedures. SQL-ін’єкція, що базується на використанні процедур, що зберігаються. Метою атаки є підвищення привілеїв, відмова в обслуговуванні та виконання віддаленої команди. Процедура - це підпрограма, доступна додатків, які звертаються до системи реляційних баз даних. Процедура фактично зберігається в словнику даних БД.

Типове використання збережених процедур включає механізми перевірки даних або контролю доступу. Крім того, збережені процедури можуть консолідувати і централізувати логіку, яка спочатку була реалізована в додатках. Велика або складна обробка, що вимагає виконання декількох операторів SQL, переміщається в збережені процедури, і всі програми викликають ці процедури. Можна використовувати вкладені збережені процедури, виконуючи одну збережену процедуру з іншої.

SQL-ін'єкції типу Stored procedures призначені для виконання збережених в БД процедур. Більша частина БД має стандартний набір процедур (крім призначених для користувача процедур), які розширюють функціональні можливості БД і дозволяють взаємодіяти з ОС. Спочатку зловмисник намагається знайти тип БД за допомогою іншого методу впровадження, такого як неприпустимі/логічно некоректні запити. Як тільки зловмисник визначить, який тип БД використовується, він спробує виконати різні процедури за допомогою впровадженого коду.

Оскільки процедура, що зберігається написана розробниками, ці процедури не роблять БД вразливою для атак з використанням SQL-ін'єкцій. Збережені процедури можуть бути уразливі для виконання віддалених команд, підвищення привілеїв, переповнення буфера і навіть забезпечення

адміністративного доступу до ОС. Якщо зловмисник вводить '; SHUTDOWN; - в поля «Ідентифікатор користувача» або «Пароль» буде згенеровано наступний код SQL:

*SELECT * from user_details, where userid = 'abc' and password = ''; error; - '*

Наведена вище команда викликає відключення бази даних.

- Alternative encodings. В цьому випадку зловмисник вводить закодований текст, щоб обійти захисні методи кодування. Зловмисники організували альтернативні методи кодування через свої введені рядки, такі як використання шістнадцятирічного, ASCII, і кодування символів Unicode. Методи сканування і виявлення в повному обсязі ефективні проти альтернативного кодування. Розглянемо наступний приклад:

*SELECT * from users, where login = ' ' and password = ' '; exec (char (Ox73687574646j776e)) '*

У наведеному вище коді використовується функція char() і шістнадцятирічне кодування ASCII. Функція char() повертає фактичні символи шістнадцятирічного кодування символів. Цей закодований рядок перетвориться в команду виключення БД при її виконанні.

Розглянемо простий веб-додаток з формою входу. Код HTML-форми наведено нижче:

< form action='reg.php' method='post'>

*<input type='text' name='login' placeholder='Логин' required>
*

*<input type='password' name='password' placeholder='Пароль' required>
*

<input type='submit' value='Зарегистрироваться'>

</form> </form>

Наведений вище лістинг HTML-коду асоціюється з формою, що приймає ім’я користувача, його пароль, а потім надсилає отримані дані для обробки за допомогою програми, що асоціюється з PHP-файлом, котрий носить назву reg.php. Для відправки даних використовується метод post.

Це означає, що значення введені користувачем в форму не

відображаються в рядку браузера з URL-адресою.

Вказані дані надсилаються на сервер у виггяді запиту:

$query = «INSERT INTO userlist („.$keys.“) VALUES („.$values.“)»;

У змінній $ keys містяться ключі з супермасива $ _POST, а в $ values - значення:

```
$keys = "";
$values = "";
$first = 1;
foreach($_POST as $key => $value) {
if($first == 0) { // Додаємо кому перед кожним пунктом, крім першого
    $keys .= ",";
    $values .= ",";
}
$keys .= $key;
$values .= "'".$value."'";
$first = 0;
}
```

Це зручно, якщо в формі дуже багато полів, але через це з'являється уразливість: якщо користувач захоче запустити SQL-ін'єкцію, то він може просто створити в формі нове поле з потрібним йому ім'ям і ввести будь-яке значення. Наприклад, можна створити поле admin і ввести значення 1. Якщо СУБД погано захищена, то хакер може зробити багато спроб, але в підсумку все ж отримати доступ до прав адміністратора.

У наведеному вище прикладі використано метод ручного створення SQL-ін’єкції атаки. Разом з тим достатньо відомі автоматизовані інструменти, які дозволяють виконувати атаки ефективніше і швидше:

- SQLSmack – дозволяє на віддаленому сервері Microsoft SQL виконувати команди, передаючи їх через збережену процедуру master..xp_cmdshell. Для цього зловмиснику повинні бути відомі ім’я користувача та пароль.

- SQLPing 2 - інструмент сканування SQL серверів, який, використовуючи

словники, робить перевірки на слабкі паролі.

- SQLMap - інструмент тестування на проникнення з відкритим вихідним кодом, який автоматизує процес виявлення та використання недоліків введення SQL та прийняття серверів баз даних. Він поставляється з потужним механізмом виявлення, безліччю функцій тестера остаточного проникнення та широким набором опцій застосування SQL-ін’єкцій.

Таким чином, узагальнена схема механізму реалізації SQL-ін’єкції складається із наступних етапів:

- Запуск SQL-ін’єкції на реалізацію.

- Визначення типу та версії СУБД для подальшого врахування відомих вразливостей такої СУБД.

- Використання даних про відомі вразливості СУБД.

- Отримання НСД за рахунок відомих вразливостей БД без несанкціонованого підвищення привілеїв користувача-зловмисника

- Визначення зловмисником найменування та привілеїв поточного користувача СУБД, від імені якого здійснюється SQL-ін’єкція.

- Несанкціоноване підвищення привілеїв користувача-зловмисника.

- Отримання НСД до БД з використанням не санкціоновано отриманих привілеїв.

Сліз зазначити, що в більшості випадків цілями отримання НСД до БД являється :

- Несанкціоноване читання даних із БД.

- Несанкціонована модифікація даних БД.

- Несанкціоноване знищення даних БД.

- Отримання парольних даних користувачів БД.

- Блокування доступу до БД.

- Несанкціоноване читання, модифікація, знищення службових даних, що забезпечують штатну експлуатацію СУБД. Це може бути, наприклад, модифікація та/або впровадження зберігаємих процедур, котрі виконують певну модифікацію даних БД.

Структура схема описаного механізму реалізації SQL-ін’єкції показана на рис. 2.1.

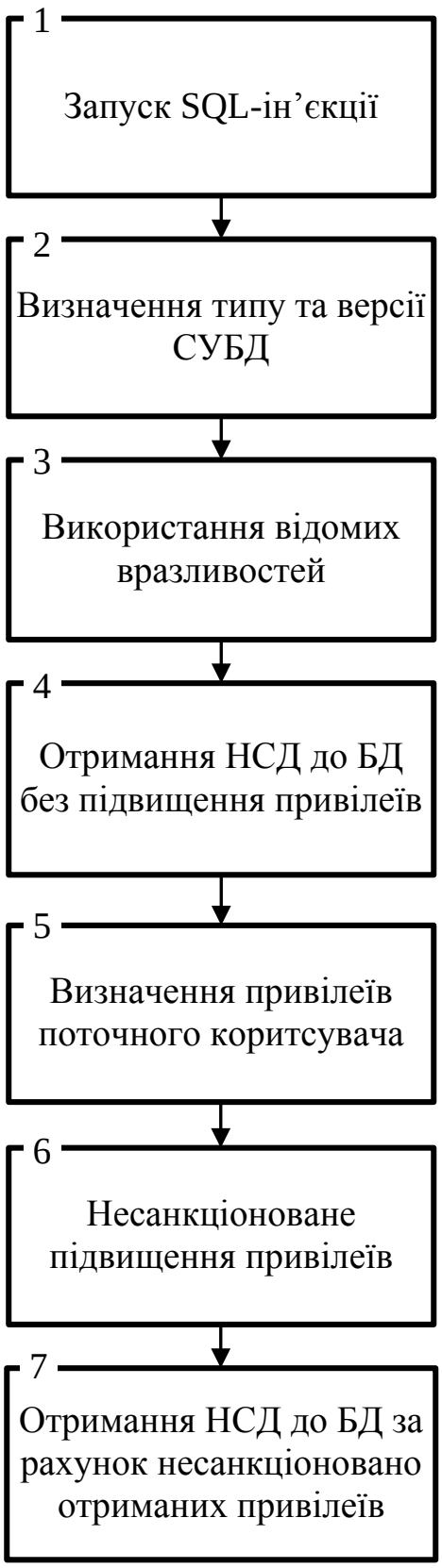

Рис. 2.1 Структурна схема узагальненого механізму реалізації SQL-ін’єкції

2.2. Процедури протидії SQL-ін’єкціям

Як відзначено в [14, 16, 19] сучасні методи протидії SQL-ін’єкціям базуються на трьох підходах, що полягають в:

1. Перевірці синтаксису клієнтських SQL-запитів.

2. Перевірці на стороні серверу ідентифікаторів та ключових слів, що використовуються в SQL-запитах.

3. Подання серверу СУБД тільки таких SQL-запитів, параметри яких знаходяться в списку дозволених. Зазначимо, що даний підхід отримав назву підходу на основі плейсхолдерів. Плейсхолдер - це звичайна змінна, яка жорстко прописана в конструкції SQL-запиту та не залежить від даних. При цьому дані від користувача надсилаються на сервер окремо від запиту, і в свою чергу від плейсхолдер не залежать. Тільки після того, як запит буде інтерпретований, дані буде використано вже безпосередньо на етапі виконання. На практиці це виглядає так: при виклику prepare() запит від користувача надсилається на сервер прямо у вигляді змінних-плейсхолдерів. Сервер реалізує обробку цих змінних і сигналізує або про успішну обробку, або виникнення помилки. А потім, при виконанні execute(), на сервер надсилаються дані клієнтського запиту. Запит надсилається в бінарному пакеті, що за структурою відповідає результату запиту. Отримані дані використовується для виконання SQL-запиту.

Відповідно до результатів першого розділу в SQL-запитах, що мають вразливості достатньо часто повинні використовуватись цілочисельні значення. Зловмисник може використати такий параметр. Тому тип параметру, що призначений для такої передачі можливо примусово привести до цілого числа. Якщо зловмисник передасть в такому параметрі замість числа SQL-код, то ружельтатом приведення буде нуль. При цьому логіка всього SQL-запиту не зміниться і збою у функціонуванні серверу СУБД не виникне.

Крім того, SQL-запити достатньо вразливі у випадку необхідності використання символьних даних, що введені користувачем. Наприклад, при введенні користувачем в форму пошуку міста по його назві. В цьому випадку

форма пошуку по протоколу GET передає на сервер додатку пошуковий змінну, що використовується в SQL-запиті:

$name = $_GET['name']

$n_sql = "SELECT * FROM citie WHERE names LIKE('%$name%')"

При цьому, якщо в параметрі name буде символ лапки, то сенс запиту можна кардинально змінити. Передавши в *'name'* значення ') + and + (id <>' 0, ми виконаємо запит, що виведе список всіх міст:

*SELECT * FROM citie WHERE names LIKE('%') AND (id<>'0%'))*

Сенс запиту змінився, тому що лапки з параметра запиту вважається керуючим символом, адже в багатьох випадках сервер СУБД визначає закінчення значення по символу лапки після нього, тому самі значення лапки містити не повинні.

Очевидно, приведення до числового типу не підходить для строкових значень. Тому, щоб убезпечити строкове значення, використовують операцію екранування.

Екранування додає в рядку перед лапками (і іншими спецсимволами) знак зворотного слеша \. Така обробка позбавляє лапки їх статусу - вони більше не визначають кінець значення і не можуть вплинути на логіку SQL-вирази.

Також результати проведеного аналізу дозволяють стверджувати, що внесення зловмисником змін в логіку виконання SQL-запиту пов’язане з необхідністю впровадження в такий запит нових достатньо довгих рядків. Тому, якщо максимальну довжину коректного значення параметра можливо визначити наперед, то одним з методів синтаксичного захисту може полягати в обмеженні максимальної довжини значень вхідних параметрів.

Таким чином для реалізації першого підходу розроблено наступні процедури перевірки синтаксису.

Процедура 1. Полягає в примусовому приведенні типу змінних, що використовуються для передачі цілочисельних даних до типу integer.

Процедура 2. Полягає в екрануванні символьних значень.

Процедура 3. Полягає у обмеженні максимальної довжини вхідних параметрів.

Розглянемо передумови процедур, що базуються на другому та третьому підходах до протидії SQL-ін’єкціям. Відповідно до першого підходу дані повинні підставлятись в SQL-запит не безпосередньо, а після певної перевірки та підстановки, що зумовлює використання проміжного шару обробки. При цьому код SQL-запиту та дані, що в нього додаються формуються окремо. Основними перевагами такого способу обробки є:

– Компактність програмного коду.

– Спрощення логіки програмної обробки SQL-запиту.

Не потрібно використовувати різноманітні алгоритми для обробки різних частин SQL-запиту. Дані оброблюються одноразово, коректно, і отримані результати потраплять строго за призначенням - в БД. Відповідно, зловмиснику більш складно вивчити систему захисту і знайти потрібні йому вразливості. Очевидно, що підстановку даних через плейсхолдер слід реалізовувати незалежно від джерела отримання інформації або будь-яких інших умов.

В багатьох випадках користувач повинен мати можливість додати в запит не тільки дані, але і інші елементи БД - ідентифікатори (імена полів і таблиць) та ключові слова. Наприклад, користувачеві надається можливість перегляду бази промислових товарів, що відображається у вигляді HTML-таблиці. При цьому користувач повинен мати змогу сортувати вказану таблицю по будь-якому з полів, в напрямку зростання/зменшення даних у цьому полі. Таким чином, користувач має змогу передавати в SQL-запит назву колонки таблиці та напрямок сортування. У випадку безпосередньої підстановки таких даних в SQL- запит небезпека реалізації SQL-ін'єкції значно підвищується, хоча і дещо нівелюється за допомогою процедур, що базуються на синтаксичній фільтрації. Крім того, проводити класичну фільтрацію ідентифікаторів не доцільно, оскільки:

– У випадку задання неправильного імені поля таблиці такий запит

призведе до повідомлення про помилку. Таке повідомлення буде доступне зловмиснику, що дозволить йому вивчати систему захисту.

– У випадку підстановки імен полів без попередньої фільтрації зловмисник може вписати в запит ті імена полів, які йому переглядати заборонено. Наприклад, у випадку реєстрації при автоматичному формуванні SQL-запит на базі масиву $_POST зловмисник може додати в форму поле admin зі значенням «1» і отримати НСД до БД.

Тому всі можливі варіанти вибору повинні бути наперед однозначно визначені. Відповідно в SQL-запит повинні потрапляти тільки такі значення. Підставою для їх вибору має бути вибір користувача.

Загальновідомі два варіанти реалізації плейсхолдерів. Серверний варіант реалізації плейсхолдера. У цьому випадку запит йде на сервер з плейсхолдеру, а дані на сервер надсилаються окремо від запиту. Обробка такого плейсхолдеру реалізується сервером СУБД. Клієнтський варіант реалізації плейсхолдера. У цьому випадку дані форматуються і підставляються в рядок запиту на безпосередньо на клієнті, формуючи класичний SQL-запит.

Врахуємо, що для більшості поширених мов програмування, котрі використовуються для розробки серверного ПЗ існують бібліотеки, що надають розробнику простий і універсальний інтерфейс для доступу до різних БД. Як правило за замовчуванням в таких бібліотеках реалізовано клієнтський варіант реалізації плейсхолдерів, що тільки емулюють серверний плейсхолдер. Однак такий функціонал можна відключити, змусивши вказані розширення надсилати на сервер дані окремо від запиту.

Розглянемо принципи обробки різних елементів SQL-запиту. Оскільки правила обробки залежать від типу елемента SQL-запиту, слід:

- Отримати інформацію про те який саме елемент SQL-запиту підлягає обробці.

- Передати отриману інформацію обробнику плейсхолдерів.

Зазначимо, що в SQL-запиті можливо виділити три основні групи елементів:

- Елементи мови SQL - оператори, вбудовані функції та змінні;

- Ідентифікатори - імена БД, таблиць і полів);

- Літерали - дані, підставлені безпосередньо в запит. Літерали в свою чергу можуть мати кількох різних типів.

Тільки останні два пункти вимагають спеціального форматування. Треба повідомити інформацію про тип даних нашого обробника.

Форматування ідентифікаторів. Взагалі, правила іменування ідентифікаторів достатньо складні. Але з огляду на те, що в системі захисту використовуються білі списками, а плейсхолдер - тільки для форматування, то достатньо взяти ідентифікатор в зворотні одинарні лапки. У випадку коли така лапка зустрічається в імені ідентифікатора, то вона також повинна бути екранована. Наприклад:

function escId($v) { return "` ".str_replace (" `", "` `", $v). "` ";}

Власне, якщо говорити про використання плейсхолдерів в цілому, саме послідовне, без винятків, застосування правил форматування і дозволяє говорити про гарантованого захисту від ін'єкцій і - що важливо - помилок.

Форматування строкових літералів. Сформулюємо правила форматування рядків в SQL:

- Рядок повинен бути взятий в лапки. Вони можуть бути одинарні або подвійні, але оскільки подвійні можуть бути використані для ідентифікаторів, краще завжди використовувати одинарні.

- В рядку повинні бути екрановані спецсимволи за списком. Для цього API більшості серверних мов програмування надає спеціальну функцію. Для коректної роботи цієї функції повинна бути правильно задана кодування з'єднання

Форматування чисел. Теоретично, в більшості випадків числа можна форматувати як рядки, і тоді завдання зведеться до попередньої. При цьому виникають наступні труднощі:

- Якщо в СУБД включено режим STRICT MODE це може призвести до видачі помилки при спробі видати рядок за число.

- Оператор LIMIT, в якому використання рядків не передбачено.

- Тип літерала дуже важливий при плануванні і виконанні запиту.

- Недостатня розрядність вбудованих в мови програмування механізмів приведення типів може призвести до необхідності додаткової обробки таких даних.

- Дійсні числа бажано перевіряти тільки регулярними виразами, оскільки в типи даних в серверних мовах програмування можуть не співпадати з типами даних в СУБД.

Таким чином в базовому варіанті запропонувати ще три процедури попередньої обробки SQL-запиту. Послідовне виконання даних процедур призводить до виконання SQL-запиту, параметри якого встановлюються за допомогою плейсхолдерів.

Процедура 4. Підготовка виразу для виконання SQL-запиту.

Процедура 5. Виконання SQL-запиту.

Процедура 6. Отримання результатів виконання SQL-запиту.

2.3 Проектування архітектури модулю захисту

Аналіз науково-практичних літературних робіт, що був проведений у розділі 1 та розробка процедур захисту від SQL-ін’єкцій дозволила перейти до проектування архітектури модулю захисту СУБД. Для опису архітектури було використано мову моделювання UML ((Unified Modeling Language)). UML була розроблена з метою забезпечити єдиний візуальну мову з багатою семантикою і розгорнутим синтаксисом для проектування і впровадження програмних систем зі складною структурою і комплексною поведінкою.

Варто відзначити, що UML застосовується не тільки в розробці програм, але і в інших сферах, наприклад, в схематизації потоків виробничих процесів. UML нагадує стандарти, що використовуються в інших галузях, і підтримує діаграми декількох типів. В цілому, діаграми UML описують межі, структуру і поведінку як всієї системи, так і окремих об'єктів, що знаходяться в її складі.

До основних переваг UML відносять:

- хороше сприйняття та виразність,

- наявність механізмів розширення і спеціалізації базових концепцій,

- нотація UML не залежить від конкретних мов програмування і інструментальних засобів проектування,

- інтеграція новітніх і найкращих досягнень в області об’єктно орієнтованої концепції проектування інформаційних систем.

При використанні UML архітектура системи представляється за допомогою діаграм, що представляють собою моделі, котрі описують частини цієї системи.

Процес проектування починається з побудови діаграми варіантів використання. Діаграма варіантів використання модулю захисту СУБД показана на рис. 2.2.

При побудові цієї діаграми враховано, що до її складу входить 3 актори «Оператор», «Клієнт» та «Сервер СУБД». Характеристики перерахованих акторів:

- «Оператор» є діючою особою, що взаємодіє з системою для встановлення її налаштувань.

- «Клієнт» представляє собою клієнтський додаток, що формує запит до СУБД та отримує від неї відповідь.

- «Сервер СУБД» представляє собою серверне апаратно-програмне забезпечення СУБД, що приймає та реалізує SQL-запити і надсилає відповіді клієнтському ПЗ для формування та відправки відповіді клієнту.

Розглянемо детальніше варіанти використання. При цьому варіанти використання відповідають функціям модуля.

Назва варіанту використання: «Задати параметри введення»

Мета варіанту використання: визначити параметри, що можуть бути використані для формування SQL-запиту.

Назва варіанту використання: «Введення даних»

Мета варіанту використання: реалізувати введення даних клієнтом для подальшого формування SQL-запиту.

Назва варіанту використання: «Задати параметри фільтрації»

Мета варіанту використання: визначити параметри, що будуть використані в процедурах обробки даних, отриманих від клієнта.

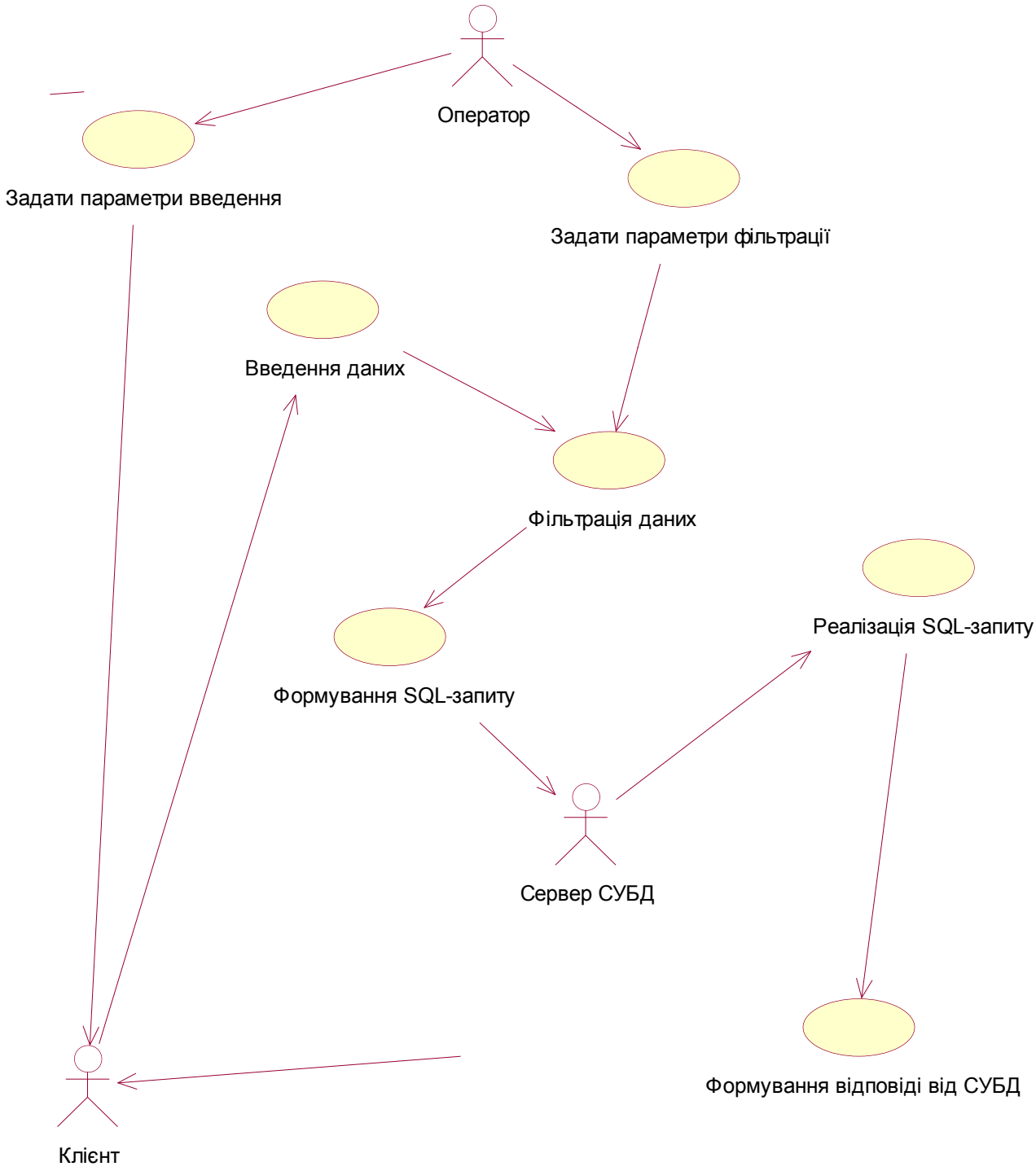

Рис. 2.2. Діаграма варіантів використання

Назва варіанту використання: «Фільтрація даних»

Мета варіанту використання: реалізація процедур обробки отриманих від

клієнта параметрів, що будуть використані при формуванні SQL-запиту.

Назва варіанту використання: «Формування SQL-запиту»

Мета варіанту використання: формування SQL-запиту, параметри якого пройшли всі етапи обробки.

Функціонал інших варіантів використання відповідає їх назвам.

На рис. 2.3 показана діаграма пакетів. На цій діаграмі відображено пакети та підпакети, з яких складається модуль розпізнавання:

– «Монітор» – пакет, що відповідає за отримання даних від клієнта та передачу клієнту даних від СУБД. До складу цього пакету входять підпакети «Отримувач» та «Передавач. Функціонал цих пакетів відповідає їх назвам.

– «Захисник» – пакет, що відповідає за аналіз обробку інформації від клієнта для забезпечення захисту від SQL-ін’єкцій. До його складу входять підпакети «Фільтратор» та «Плейсхолдер. Перший підпакет відповідає за синтаксичну обробку запитів від клієнта. Другий підпакет відповідає за фільтрацію даних від клієнта, що реалізується за допомогою плейсхолдерів

– «Модератор» – пакет, що відповідає за налаштування системи захисту. Цей пакет складається із двох підпакетів «Адміністратор» та «Сигналізація». Перший підпакет призначений для надання оператору системи захисту можливості налаштування параметрів захисту. Другий пакет забезпечує сигналізацію щодо налаштувань системи захисту.

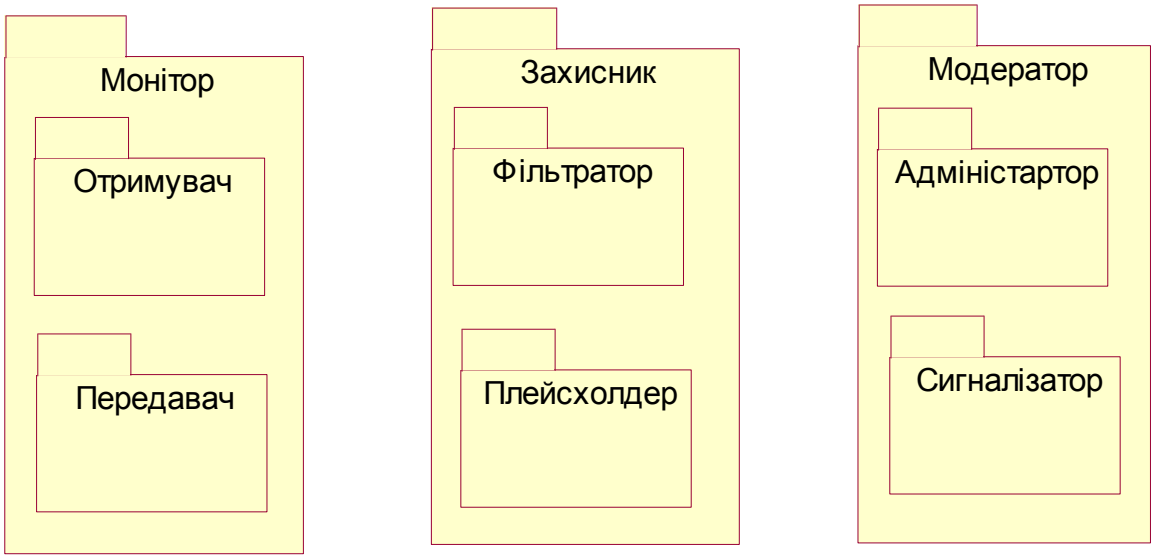

Рис. 2.3. Діаграма пакетів

Діаграма класів, що використовується для відображення структур класів, які складають архітектуру модулю захисту СУБД, показана на рис. 2.4.

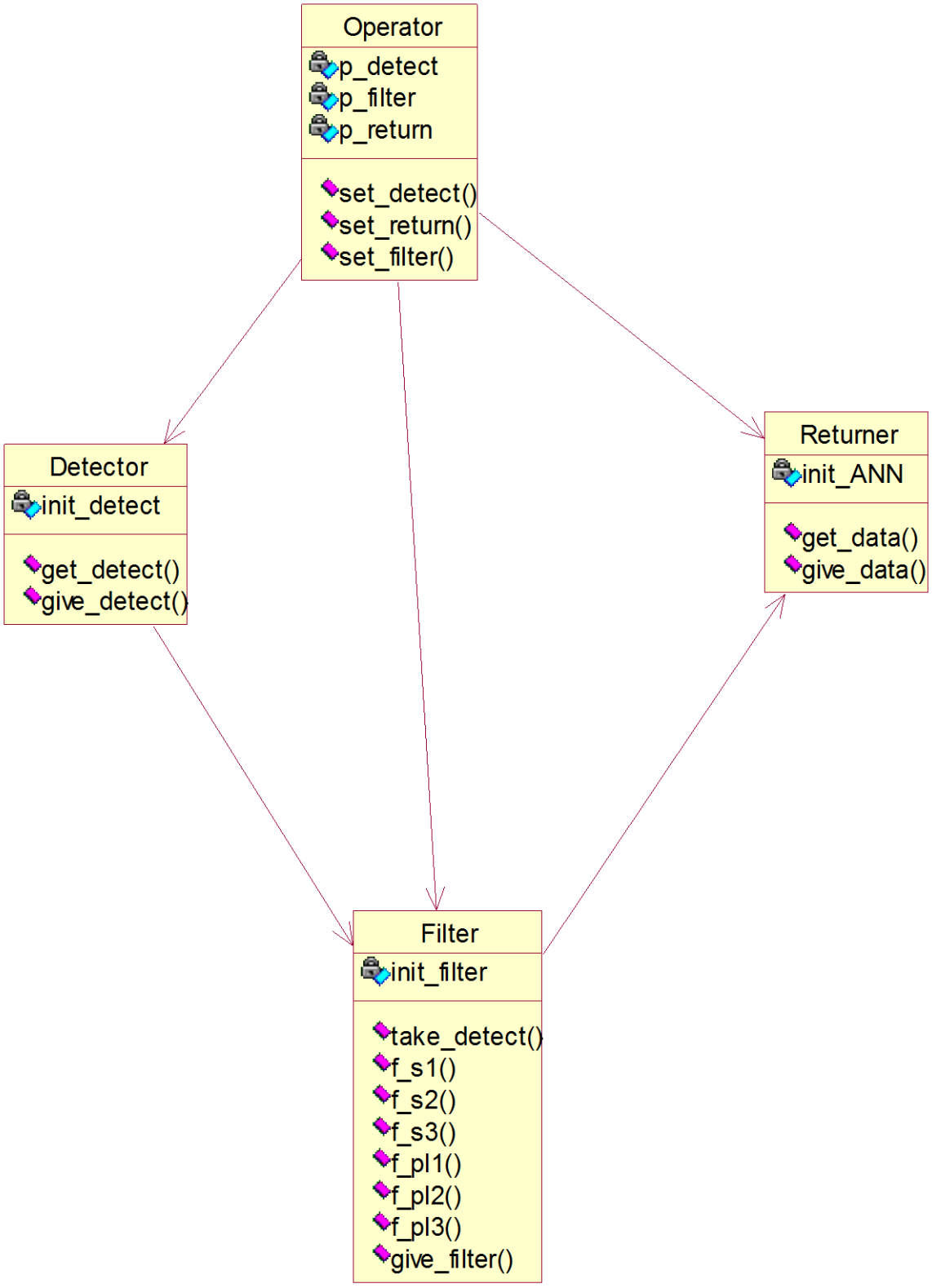

Рис. 2.4. Діаграма класів

При побудові цієї діаграми враховано, що до її складу входить 4 класи:

- «Operator» - відповідає за визначення базових налаштувань модулю захисту,

- «Detector» - відповідає за отримання даних від клієнта,

- «Returner» - відповідає за відправку даних клієнту,

- «Filter» - відповідає за фільтрацію даних, отриманих від клієнта.

Класи пов’язані між собою пов’язані між собою відношенням залежності.

Наведемо коротку характеристик атрибутів та методів вказаних класів.

Атрибути класу «Operator» призначені для:

- p_detect – зберігання налаштувань щодо «Detector»,

- p_filter –зберігання налаштувань щодо «Filter»,

- p_return – зберігання налаштувань щодо «Returner»,

Методи класу «Operator» призначені для :

- set_detect() – призначений для передачі налаштувань «Detector»,

- set_filter() – призначений для передачі налаштувань «Filter»,

- set_return() – призначений для передачі налаштувань «Returner».

Атрибут init_detect класу «Detector» призначений для зберігання налаштувань зчитування даних від клієнта.

Методи класу «Detector» призначені для:

- get_detect() – отримання даних від клієнта,
- give_detect() – передачі даних отриманих від клієнта до класу фільтрації.

Атрибут init_ANN класу «Returner» призначений для зберігання налаштувань відправки даних від СУБД до клієнта.

Методи класу «Returner» призначені для:

- get_data() – отримання даних від класу фільтрації,
- give_data() – передачі даних отриманих від класу фільтрації клієнту.

Атрибут init_filter класу «Filter» призначений для зберігання налаштувань відправки даних від СУБД до клієнта.

Методи класу «Filter» призначені для:

- take_detect() – отримання даних від клієнта,

- give_filter() – передачі відфільтрованих даних класу «Returner».

- f_s1() – екрануванні символьних значень,

- f_s2() – приведення даних до числового типу,

- f_s3() – обмеження довжини запиту,

- f_pl1() – підготовці до виконання SQL-запиту за рахунок підстановки даних з використанням плейсхолдерів,

- f_pl2() – виконання SQL-запиту,

- f_pl3() – отримання результатів виконання SQL-запиту.

На рис. 2.5 показана діаграма компонентів, що складається із трьох модулів: «Detecor», «Filter» та «Admin».

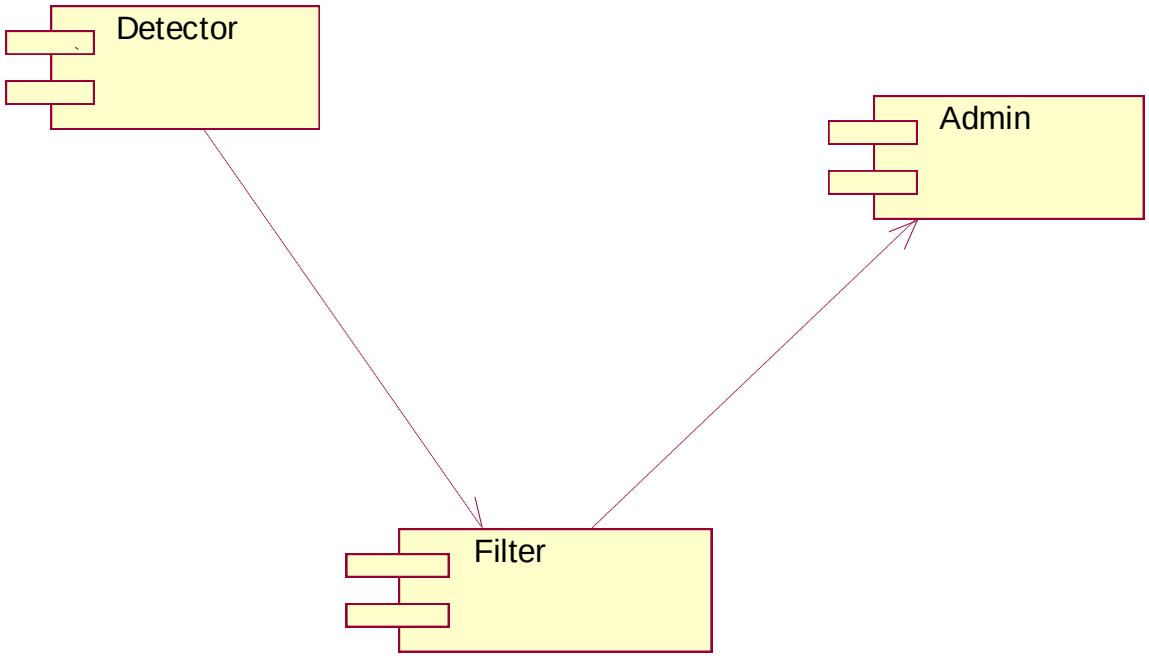

Рис. 2.5. Діаграма компонентів

Вказані модулі визначають інтерфейс отримання даних для формування SQL-запиту, фільтрацію вказаних даних та налаштування системи захисту.

2.4. Висновки до розділу 2

Другий розділ присвячено вирішенню задачі захисту реляційних баз даних. В результаті проведених досліджень:

- Розроблено формалізований опис механізмів SQL-ін’єкцій. Показано, що типи атак, які можуть бути виконані з використанням SQL-ін'єкції,

розрізняються за типом механізмів бази даних, що підлягають враженню. Визначено основні класи SQL-ін'єкцій. Побудована структурна схема узагальненого механізму реалізації SQL-ін’єкції. Наведено особливості механізму реалізації SQL-ін'єкцій кожного із визначених класів.

- З урахуванням формалізованого опису механізмів реалізації SQL-ін'єкцій розроблено процедури протидії SQL-ін'єкціям. Процедури забезпечують перевірку синтаксису клієнтських SQL-запитів, перевірку на стороні серверу ідентифікаторів та ключових слів, що використовуються в SQL-запитах та подання серверу СУБД тільки таких SQL-запитів, параметри яких знаходяться в списку дозволених.

- З урахуванням розроблених процедур протидії з використанням мови проектування UML створено діаграми модулю захисту від SQL-ін’єкцій. Побудовані UML-діаграми прецендентів, пакетів, компонентів та класів, що забезпечило деталізований опис та документування модулю захисту реляційних баз даних від SQL-ін’єкцій з орієнтацією на його наступну реалізацію у вигляді програмного забезпечення.

Результати проведених досліджень забезпечили можливість розробки програмного забезпечення модулю захисту реляційних баз даних.

Розділ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вибір засобів програмування та середовища розробки

Для реалізації системи захисту використано мову програмування PHP (англ. PHP: Hypertext Preprocessor - «PHP: препроцесор гіпертексту»). Вибір мови програмування пояснюється тим, що PHP це мова програмування, що довгий час являється однією із найбільш поширених мов в розробці серверної частини веб-додатків. Як відзначено в [18], в даний час PHP є одним з лідерів серед серверних мов програмування, що застосовуються для створення динамічних веб-сайтів і веб-додатків. Велика частина коробкових систем управління сайтами написана саме на PHP, мова підтримується переважною більшістю хостинг-провайдерів. Мова набла широкого поширення завдяки своїй простоті, швидкості, мультипарадигмальності, багатій функціональності і кросплатформеності.

PHP - мова програмування, створена для генерування HTML-сторінок на веб-сервері і роботи з БД. В даний час підтримується переважною більшістю хостинг-провайдерів. Входить в LAMP - поширений набір для створення веб-сайтів (Linux, Apache, My SQL, PHP (Python або Perl)).

Код PHP може об'єднуватися з тегами XHTML. PHP є вбудованою мовою програмування. Це означає, що код PHP можна перемішуватись з кодом HTML і PHP, не жертвуючи можливістю читання тексту. Від HTML-коду код PHP відділяється спеціальними початковим і кінцевим тегами <? Php і ?>, що дозволяють "перемикатися" в "PHP-режим" і виходити з нього. Однак хорошим стилем програмування є відділення програмного коду на мові PHP від подання до вигляді HTML. Це найчастіше реалізується за допомогою шаблонізації. PHP - інтерпретована мова програмування, що дозволяє створювати програми в процедурному і об'єктно-орієнтованому стилі.

Для розробки ПЗ використано фреймворк Yii.

Yii - це популярний фреймворк для php-розробки, заснований на парадигмі MVC. Основна перевага Yii - дуже висока швидкість роботи і, як наслідок, продуктивність.

Фреймворк активно розвивається спільнотою. Yii не виглядає «монстром» в порівнянні з фреймворками Symfony і Zend Framework, кодова база яких вельми об'ємна. Фреймворк Yii досить простий в освоєнні і у використанні, що сприяє швидкій розробці на ньому проектів. Як і всі фреймворки, Yii «заточений» під розробку технічно складних веб-проектів: бізнес-додатків, веб-сервісів, а також сайтів зі складною бізнес-логікою і вимогливих до швидкодії.

Основні переваги і можливості фреймворка Yii:

- Забезпечує високу продуктивність щодо інших php-фреймворків.

- Заснований на парадигмі MVC (Модель-Представлення-Контролер).

- Є інтерфейси DAO і ActiveRecord для роботи з базами даних (використовується PDO).

- Підтримує інтернаціоналізацію.

- Дозволяє кешувати як сторінки цілком, так і окремі фрагменти.

- Здійснює перехоплення і обробку помилок.

- Має функціонал роботи з формами, забезпечує їх побудову та валідацію.

- Реалізовані аутентифікація і авторизація.

- Зручний для реалізації AJAX-інтерфейсів, інтегрується з jQuery.

- У фреймворк вбудовані генератори базового PHP-коду для CRUD-операцій (скаффолдинг).

- Підтримує теми оформлення.

- Має можливість підключення сторонніх бібліотек.

- Працює з міграціями баз даних (генерація, застосування і відкат).

- Дозволяє здійснювати автоматичне тестування і вести розробку в стилі TDD. Зазначимо, що TDD (test-driven development або процес розробки через тестування) - це методологія розробки ПЗ, яка ґрунтується на повторенні коротких циклів розробки: спочатку пишеться тест, що покриває бажану зміну, потім пишеться програмний код, який реалізує бажане поведінку системи і

дозволить пройти написаний тест, а потім проводиться рефакторинг написаного коду з постійною перевіркою проходження всіх тестів.

- Підтримує стиль REST. REST - це стиль побудови архітектури розподіленого клієнт-серверного додатка, який спрощує роутинг і побудову API. REST є дуже простим інтерфейсом управління інформацією без використання додаткових внутрішніх прошарків, тобто передача даних здійснюється без надмірних «обгорток». Кожен об'єкт однозначно визначається глобальним ідентифікатором, таким як URL, а кожен URL має строго заданий формат. Управління цими ресурсами здійснюється за допомогою стандартного інтерфейсу, наприклад, через HTTP, а обмін інформацією відбувається за допомогою уявлень цих ресурсів.

3.2. Загальний опис програмного модулю

Базуючись на побудованих UML-діаграмах побудовано узагальнений алгоритм роботи ПЗ. Структура даного алгоритму показана на рис. 3.1.

Відмінними рисами ПЗ є реалізація наведених в розділі 2 процедур, що забезпечують обробку запитів від клієнтів.

Процедура 1. Реалізація цієї процедури полягає у застосуванні функції PHP для примусового приведення змінної до цілочисельного типу. Наприклад,

$city_id = $_GET['city_id'];

settype($city_id, 'integer');

Після такого приведення змінна $city_id може використовуватись в SQL-запитах.

Процедура 2. Реалізація цієї процедури полягає у застосуванні функції PHP для екранування символьних значень. Наприклад,

$city_name = mysqli_real_escape_string($link, $_GET['search']);

$sql = "SELECT * FROM cities WHERE name LIKE('%$city_name%')";

В наведеному програмному коді реалізована обробка змінної $city_name, що передається від клієнта на сервер по протоколу GET.

Процедура 3. Реалізація цієї процедури полягає у застосуванні функції

PHP для обмеження довжини рядка даних, що використовується для формування SQL-запиту. Наприклад, якщо відомо, що поле id може приймати значення не більше 9999, можна «відрізати зайві» символи, залишивши не більше чотирьох:

*statement:= 'SELECT * FROM users WHERE id = ' + LeftStr(id, 4) + ';'*

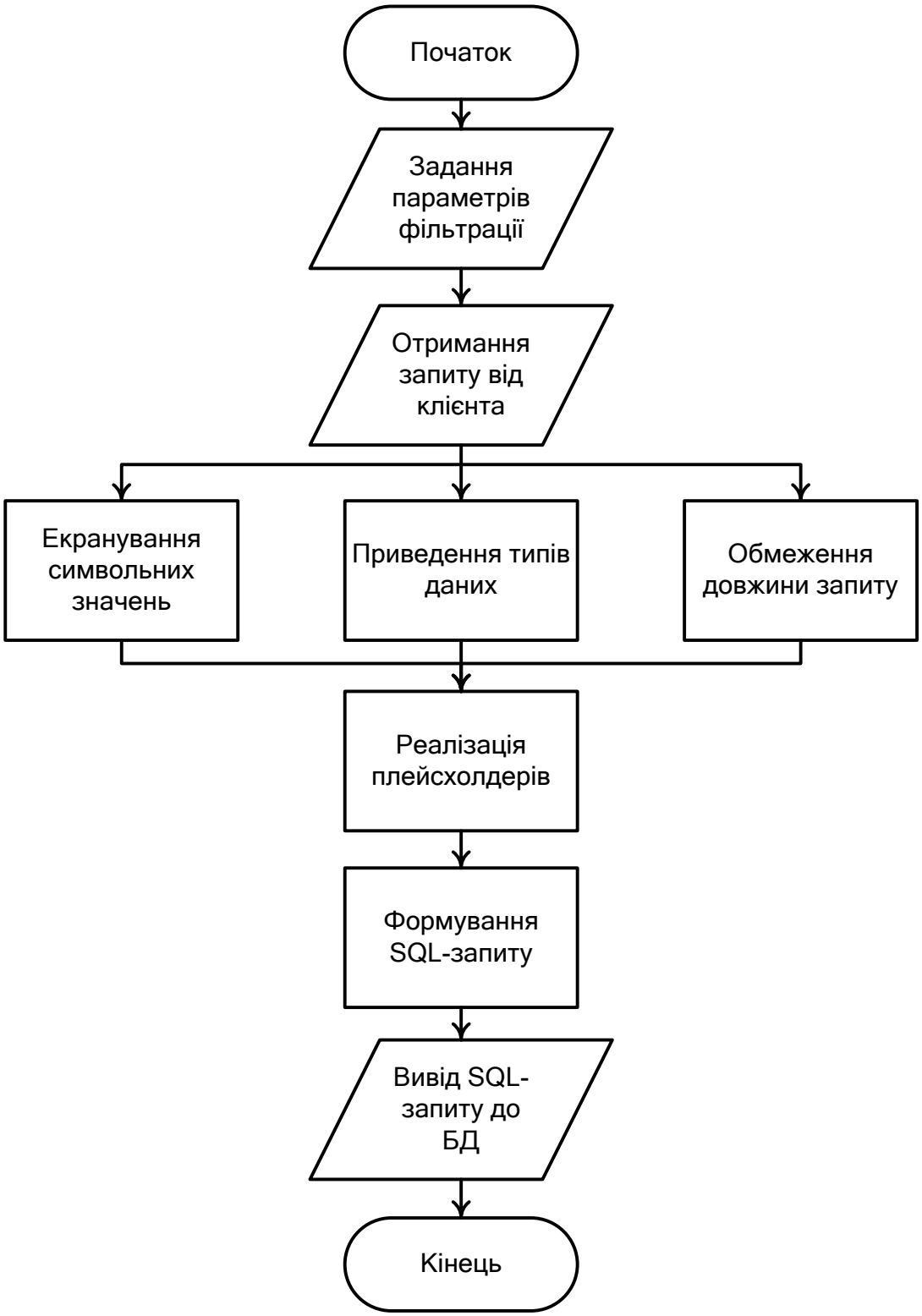

Рис. 3.1. Структура алгоритму функціонування програмного модулю захисту баз даних від ін’єкцій

Процедура 4. Реалізація цієї процедури полягає у підготовці SQL-запиту, що зводиться до застосування плейсхолдерів. Ці плейсхолдери надалі будуть замінені на реальні значення. Шаблон запиту відправляється на сервер СУБД

для аналізу і синтаксичної перевірки. Наприклад,

*$sql = "SELECT * FROM cities WHERE name = ?";*

$stmt = mysqli_prepare($link, $sql);

Процедура 5. Реалізація цієї процедури полягає у тому, що під час запуску запиту PHP прив'язує до плейсхолдеру реальні значення і посилає їх на сервер. За передачу значень в підготовлений запит відповідає функція mysqli_stmt_bind_param(). Вона приймає тип і самі змінні. Наприклад,

mysqli_stmt_bind_param($stmt, 's', $_GET['search']);

Процедура 6. Реалізація цієї процедури полягає у тому, щоб після виконання SQL-запиту отримати його результат. Для СУБД MySQL результат виконання SQL-запиту має формат mysqli_result. Для його отримання можна скористатись функцією mysqli_stmt_get_result(). Наприклад,

$res = mysqli_stmt_get_result($stmt);

```
// читання даних
while ($row = mysqli_fetch_assoc($res)) {
   // асоціативний масив з черговим рядком результату виконання
   var_dump($row);
}
```

Тестування розробленого програмного модулю захисту реляційних БД було проведено шляхом його впровадження в систему захисту веб-сайту Інтернет-магазину. Зазначимо, що розроблений модуль призначений для експлуатації на веб-серверах, котрі можуть забезпечувати роботу інтерпретатора PHP 5.1 та вище.

Орієнтовні апаратні вимоги при одночасній роботі 5 користувачів з таблицями до 1 млн рядків:

- оперативна пам'ять - 2 + ГБ,
- вільне місце на жорсткому/SSD диску: 30+ ГБ,
- частота процесора: 1+ ГГц x 2

Установка можлива на будь-яку операційну систему, включаючи Windows, Linux, MacOS. При тестуванні розроблений модуль захисту був

розгорнутий на ОС Ubuntu Server. Функціонування Інтернет-магазину забезпечував веб-сервер Apache у поєднанні з інтерпретатором PHP та СУБД MySQL.

Тестові подачі SQL-ін’єкцій, що були реалізовані за рахунок некоректного синтаксису та некоректної підстановки даних не призвели до НСД до даних БД. Зловмиснику надсилались відповіді у вигляді форми для повторного введення запиту. Для прикладу на рис. 3.2 та рис.3.3 наведено форми невдалого та вдалого входу до Інтернет-магазину від імені неіснуючого користувача, що були реалізовані з використанням SQL-ін’єкцій. При цьому вдалий вхід до Інтернет-магазину був здійснений у випадку відсутності розробленого модулю захисту, а невдалий вхід – у випадку функціонування запропонованого модулю.

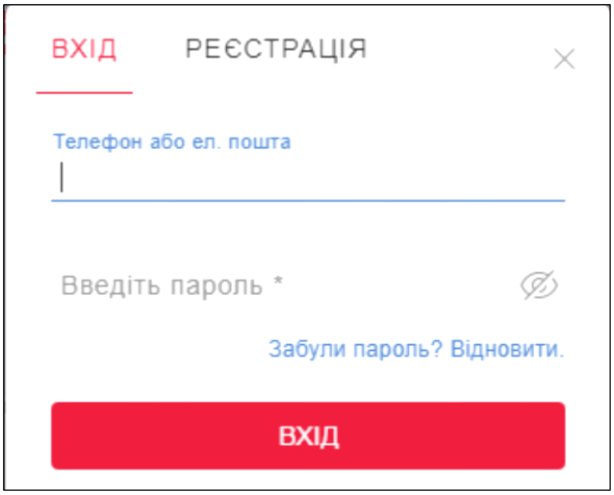

Рис. 3.2 Ілюстрація невдалого входу до Інтернет-магазину неіснуючим користувачем

При цьому експлуатаційні властивості Інтернет-магазину не було порушено. Тобто веб-сайт Інтернет-магазину вчасно відповідав на коректні запити легітимних користувачів. Таким чином, показана доцільність застосування розробленого модулю у комплексних системах захисту Інтернет-орієнтованих ІС.

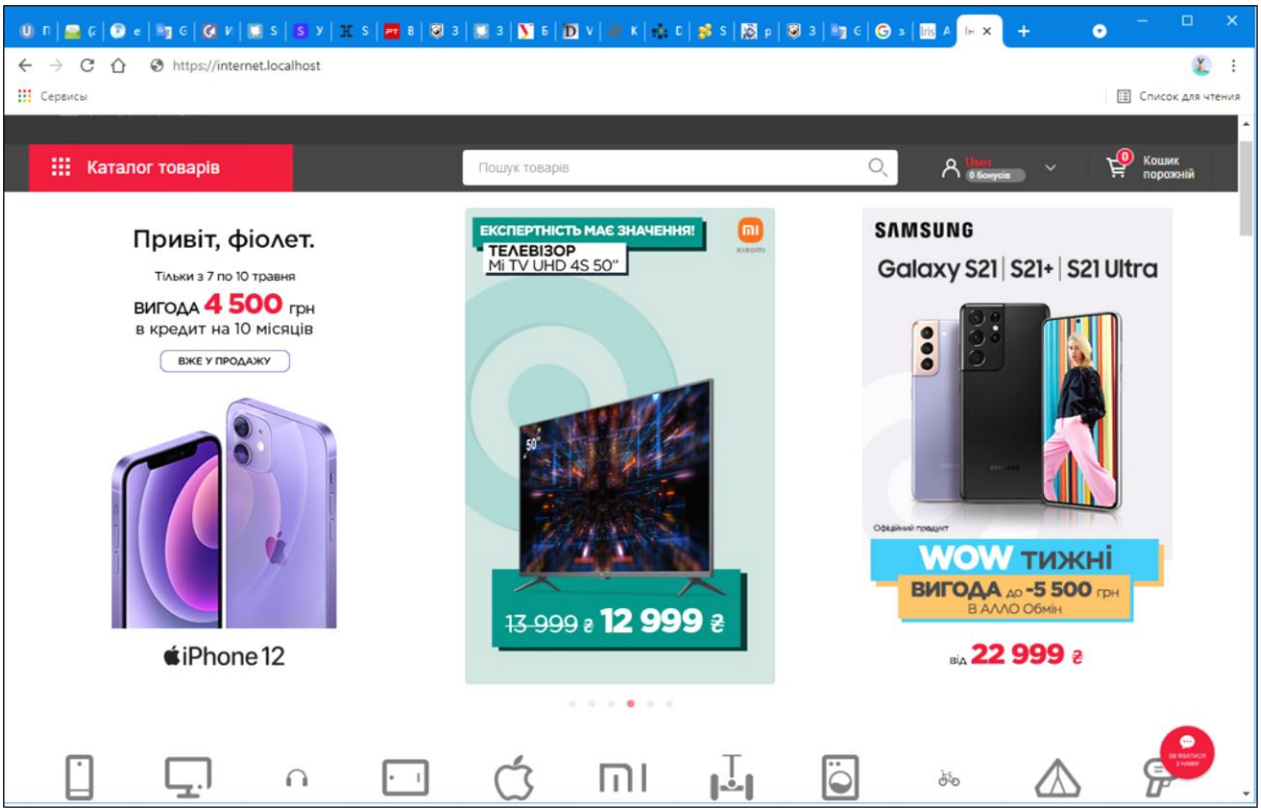

Рис. 3.3 Ілюстрація вдалого входу до Інтернет-магазину неіснуючого користувача User

.

3.4. Висновки до розділу 3

Третій розділ присвячено вирішенню задачі розробки та тестування програмного забезпечення модулю захисту реляційних баз даних. В результаті проведених досліджень:

- Обґрунтовано вибір технологій розробки програмного забезпечення. Показана доцільність використання мови PHP та фреймворку Yii.

- Розроблено програмний модуль, що забезпечує можливість захисту реляційних баз даних від SQL-ін’єкцій.

- Тестування розробленого модулю у складі системи захисту Інтернет-магазину було здійснене шляхом подачі SQL-ін’єкцій, що були реалізовані за рахунок некоректного синтаксису та некоректної підстановки даних. Результати тестування показали, що такі SQL-ін’єкції не призвели до несанкціонованого доступу до БД. Таким чином, показана доцільність

застосування розробленого модулю у комплексних системах захисту Інтернет-орієнтованих ІС.

ВИСНОВКИ

Дипломна робота присвячена розв’язанню задачі розробки програмного модулю захисту реляційних баз даних. В процесі розв’язання отримано наступні результати:

1. Визначено, що один із пріоритетних напрямків вдосконалення систем захисту реляційних баз даних пов’язаний із вдосконаленням технології протидії SQL-ін’єкціям. Показано, що вдосконалити технологію протидії SQL-ін’єкціям можливо за рахунок розробки програмного модулю.

2. Здійснена формалізація механізмів реалізації SQL-ін’єкцій, що забезпечило можливість визначення шляхів протидії, котрі базуються на обробці запитів від клієнта.

3. Розроблено процедури протидії найбільш розповсюдженим SQL-ін’єкціям, що базуються на спеціально спотвореному синтаксисі запитів від клієнта та на некоректній підстановці даних у ці запити.

4. Розроблена архітектура програмного модулю захисту реляційних баз даних від SQL-ін’єкцій, що забезпечило базис розробки програмного додатку.

5. Розроблено програмний модуль, що забезпечує можливість захисту реляційних баз даних від SQL-ін’єкцій.

6. Тестування розробленого модулю у складі системи захисту Інтернет-магазину було здійснене шляхом подачі SQL-ін’єкцій, що були реалізовані за рахунок некоректного синтаксису та некоректної підстановки даних. Результати тестування показали, що такі SQL-ін’єкції не призвели до несанкціонованого доступу до реляційних баз даних.

7. Перспективи подальшого розвитку даної роботи можуть бути пов’язані з вдосконаленням системи протидії SQL-ін’єкціям за рахунок застосування нейромережевих моделей та методів їх виявлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Баранчиков А.И., Баранчиков П.А., Пилькин А.Н. Алгоритмы и модели доступа к записям БД. М.: Горячая линия-Телеком, 2011. 182 с.

2. Бирюков А.А. Информационная безопасность: защита и нападение. М. : ДМК Пресс, 2012. 474 с.

3. Бортовчук Ю.В., Крылова К.А., Ермолаева Л.В. Информационная безопасность в современных системах управления базами данных // Современные проблемы экономического и социального развития. 2010. № 6. С. 224-225.

4. Горбачевская Е.Н., Катьянов А.Ю., Краснов С.С. Информационная безопасность средствами СУБД Oracle // Укр. ВУиТ. 2015 № 2 (24). С. 72- 85.

5. Егоров М. Выявление и эксплуатация SQL-инъекций в приложениях // Защита информации. INSIDE. 2011. № 2. С. 2–8.

6. Евтеев Д. SQL Injection от А до Я [Електронний ресурс] // Режим доступа: https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/PT-devteev-AdvancedSQL-Injection.pdf.

7. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. - 3-е издания: пер. с англ. / Т. Коннолли, К. Бегг. - М .: Вильямс, 2003. - 1440 с.

8. Корченко А. Нейросетевые модели, методы и средства оценки параметров безопасности Интернет-ориентированных информационных систем: монографія / А. Корченко, И. Терейковский, Н. Карпинский, С. Тынымбаев. – К. : ТОВ «Наш Формат». – 2016. – 275 с. НМОПБ

9. Корченко О. Методологія розроблення нейромережевих засобів інформаційної безпеки Інтернет-орієнтованих інформаційних систем / О. Корченко, І. Терейковський, А. Білощицький. – К. : ТОВ «Наш Формат». – 2016. – 249 с.

10. Корченко О. Г. Верифікація нейромережевих методів розпізнавання кібератак / О. Г. Корченко, І. А. Терейковський, С. В. Казмірчук // Науково-

технічний збірник «Управління розвитком складних систем» Київського національного університету будівництва і архітектури. – 2014. – Випуск 17. – С. 168-172.

11. Кузнецов С.Д. Базы данных. М.: Академия, 2012. 496 с.

12. Отношения классов – от UML к коду [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/9qELSU.

13. Полтавцева М.А. Задача хранения прав доступа к данным в СУБД на примере Microsoft SQL Server // Актуальные направления фундаментальных и прикладных исследований: матер. V Междунар. научно-практич. конф. 2015 С. 118-120.

14. Полтавцева М.А., Зегжда Д.П., Супрун А.Ф. Безопасность баз данных: учеб. помощь. СПб: Изд-во СПБП, 2015. 125 с.

15. Поляков А.М. Безопасность Oracle глазами аудитора: нападение и защита. М.: ДМК Пресс, 2014. 336 с.

16. Потапов А.Е., Манухина Д.В., Соломатина А.С., Бадмаев А.И., Яковлев А.В., Нилова А.С. Безопасность локальных баз данных на примере SQL Server Compact // Укр. Тамбов. ун-та. Серия: Естественные и технические науки. 2014. № 3. С. 915-917.

17. Смирнов С.Н. Безопасность систем баз данных. М.: Гелиос АРВ, 2017. 352 с.

18. Ткаченко Н.А. Реализация монитора безопасности СУБД MySQL в dbf / dam системах // ПДМ. Дополнение. 2014. № 7. С. 99-101.

19. Burtescu E. Database security - attacks and control methods. Journ. of Applied Quantitative Methods 2009, vol. 4, no. 4, pp. 449-454.

20. Jones M. Fight against SQL injection attacks [Електронний ресурс] // IBM. Режим доступа: https://www.ibm.com/developerworks/security/library/se-sqlinjection-attacks/index.html.

21. Murray M.C. Database security: what students need to know. JITE: IIP, vol. 9, 2010 pp. 61-77.

22. Qiu M., Davis S. Database security mechanisms and implementation. IACIS, Issues in Inform. Syst. 2002 vol. 03 pp. 529-534.

23. Rohilla S., Mittal P.K. Database Security: Threads and Challenges. Intern. Journ. of Advanced Research in Computer Science and Software Engineering, 2013, vol. 3, iss. 5, pp. 810-813.

24. Sandhu Ravi S., Jajodia Sushil. Data and database security and controls. Handbook of Information Security Management, Auerbach Publishers, 2013, pp. 181-199.

25. SQL Injection Attacks Are Rampant: How to Stop Your Next Hack Attack [Електронний ресурс]. – Режим доступу: https://goo.gl/RxnbWp.

26. SQL-инъекция [Електронний ресурс]. – Режим доступу: https://goo.gl/UEB2FN.

27. Wassermann G. Static Checking of Dynamically Generated Queries in Database Applications / Wassermann, G; Gould, C; Su, Z, et al. // ACM Transactions on Software Engineering and Methodology. – 2017. – Vol. 16, Iss. 4. – 10 p.

ДОДАТОК А

```
//Функція для безпечного формування SQL-запиту на основі даних,
//отриманих  від клієнта
function dbget() {
 /*
 usage: dbget($mode, $query, $param1, $param2,...);
 $mode - "dimension" of result:
 0 - resource
 1 - scalar
 2 - row
 3 - array of rows
 */
 $args = func_get_args();
 if (count($args) < 2) {
  trigger_error("dbget: too few arguments");
  return false;
 }
 $mode  = array_shift($args);
 $query = array_shift($args);
 $query = str_replace("%s","'%s'",$query);
 foreach ($args as $key => $val) {
  $args[$key] = mysql_real_escape_string($val);
 }
 $query = vsprintf($query, $args);
 if (!$query) return false;
 $res = mysql_query($query);
 if (!$res) {
  trigger_error("dbget: ".mysql_error()." in ".$query);
  return false;
```

```
  }
  if ($mode === 0) return $res;
  if ($mode === 1) {
    if ($row = mysql_fetch_row($res)) return $row[0];
    else return NULL;
  }
  $a = array();
  if ($mode === 2) {
    if ($row = mysql_fetch_assoc($res)) return $row;
  }
  if ($mode === 3) {
    while($row = mysql_fetch_assoc($res)) $a[]=$row;
  }
  return $a;
}
?>

// Приклади використання функції dbget()
// якщо потрібне тільки ім’я
$name = dbget(1,"SELECT name FROM users WHERE id=%d",$_GET['id']);

// якщо потрібен весь рядок
$user = dbget(2,"SELECT * FROM users WHERE id=%d",$_GET['id']);

// якщо потрібен масив
$sql  = "SELECT * FROM news WHERE title LIKE %s LIMIT %d,%d";

$news = dbget(3,$sql,"%$_GET[search]%",$start,$per_page);
```

ДОДАТОК В