

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Направление подготовки: 10.03.01 Информационная безопасность

Образовательная программа: Информационная безопасность

Дисциплина:
«Информационная безопасность баз данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
«Защита базы данных»

Выполнил студент(ы):

группа/поток: N3347 / ИББД. N63 1.5



Чу Ван Доан /

ФИО

Подпись

Проверил:

Салихов Максим Русланович /

ФИО

Подпись

Отметка о выполнении (один из вариантов:
отлично, хорошо, удовлетворительно, зачтено)

Дата

Санкт-Петербург
2024 г.

СОДЕРЖАНИЕ

Цель работы	3
Задание	4
Ход Работы	5
1. Задачи по мониторингу БД:	5
1.1 Создайте таблицу-лог, отдельную от ваших основных сущностей БД.	5
1.2 Создание триггерной функции	5
1.3 Проверка системы логирования	6
2. Задачи по шифрованию данных.	7
2.1 Создание таблицы для хранения конфиденциальных данных	7
2.2 Шифрование данных	7
2.3 Создание ключа шифрования из пароля	7
2.4 Шифрование данных	8
2.5 Доказательство расшифровки данных	8
3. Задачи по разграничению доступа в БД:	8
3.1 Создание роли	8
3.2 Назначение привилегий	9
3.3 Доступ к таблицам логов	9
3.4 Проведение проверки прав доступа	9
Вывод	11

Цель работы

Получение навыков созданию примитивных систем мониторинга, разграничения доступа и шифрования средствами СУБД.

Задание

1. Задачи по мониторингу БД:

- Создайте таблицу-лог, отдельную от ваших основных сущностей БД.
- Создайте для каждой основной таблицы в вашей БД триггер, который срабатывает при любых изменениях в БД (вставка новых данных, изменение существующих записей, удаление кортежей из таблицы). При срабатывании триггер должен вносить в таблицу-лог информацию о том, когда было произведено изменение, со стороны какой роли поступил запрос, какие кортежи поменялись, старые и новые значения.
- Прдемонстрируйте работу системы логирования для различных операций и отношений.

2. Задачи по шифрованию данных.

- Создайте таблицу с секретными данными, отдельно от ваших основных сущностей. Например, это может быть таблица с токенами или ключами доступа, для каждого класса-пользователей.
- Зашифруйте содержимое данной таблицы, в качестве алгоритма шифрования используйте любой симметричный алгоритм шифрования. Ключ шифрования для данной таблицы не должен храниться в ИС. Ключ шифрования может быть получен из индивидуального пароля для дешифрования суперпользователя (пароль не связан с паролем для входа в СУБД). Индивидуальный пароль суперпользователя и ключ шифрования может быть связан через одностороннюю функцию. Например, пусть индивидуальный пароль комбинация «!stroNgpsw31234», считаем от данного пароля детерминированную хэш-функцию (например, sha-256), полученный хэш-используем как ключ шифрования/дешифрования для симметричного алгоритма шифрования таблицы с секретными данными (например, для AES-256)
- Демонстрируем, что даже обладая полными правами администратора, но без знания индивидуального пароля невозможно получить содержимое таблицы с секретными данными

3. Задачи по разграничению доступа в БД:

- Создайте в СУБД как минимум 2 роли (суперпользователь не считается) для каждого из классов потребителей информации;
- С помощью внутренних инструментов СУБД для каждой роли определите набор привилегий по отношению к таблицам вашей БД. Руководствуйтесь принципом минимальных привилегий, если определенному классу потребителей не нужен доступ к определенным таблицам/атрибутам (список задач БД, составленный в рамках 1 ЛР), то доступ к этим таблицам/атрибутам не предоставляется. Разграничиваем доступ к представлениям, созданным в 1 ЛР, а также таблицам логирования (таблицы логирования может просматривать только суперпользователь)
- Прдемонстрируйте работу вашей системы разграничения доступа. Зайдите за каждую из ролей и покажите доступные со стороны каждой роли отношения.

Ход Работы

1. Задачи по мониторингу БД:

1.1 Создайте таблицу-лог, отдельную от ваших основных сущностей БД.

- Чтобы контролировать базу данных, мы создаем таблицу логов для записи изменений в базе данных. Используем триггеры для автоматического обновления логов каждый раз, когда выполняются операции добавления, изменения или удаления в основных таблицах.

```
CREATE TABLE main_log (  
    log_id SERIAL PRIMARY KEY,  
    operation_type TEXT,  
    operation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    user_operator TEXT,  
    changed_data JSON  
);
```

1.2 Создание триггерной функции

- Функция logging() ниже будет записывать информацию в таблицу main_log.

```
CREATE OR REPLACE FUNCTION logging() RETURNS TRIGGER AS $logging$  
BEGIN  
    IF (TG_OP = 'DELETE') THEN  
        INSERT INTO main_log (operation_type, user_operator, changed_data)  
        VALUES ('D', current_user, row_to_json(OLD));  
    ELSIF (TG_OP = 'UPDATE') THEN  
        INSERT INTO main_log (operation_type, user_operator, changed_data)  
        VALUES ('U', current_user, row_to_json(NEW));  
    ELSIF (TG_OP = 'INSERT') THEN  
        INSERT INTO main_log (operation_type, user_operator, changed_data)  
        VALUES ('I', current_user, row_to_json(NEW));  
    END IF;  
    RETURN NULL;  
END;  
$logging$ LANGUAGE plpgsql;
```

- Примените триггер к таблице Product.

```
CREATE TRIGGER product_logging  
AFTER INSERT OR UPDATE OR DELETE ON Product  
FOR EACH ROW EXECUTE FUNCTION logging();
```

- Примените триггер к таблице Bill.

```
CREATE TRIGGER bill_logging
```

```
AFTER INSERT OR UPDATE OR DELETE ON Product
FOR EACH ROW EXECUTE FUNCTION logging();
```

- Примените триггер к таблице Orders.

```
CREATE TRIGGER order_logging
AFTER INSERT OR UPDATE OR DELETE ON Product
FOR EACH ROW EXECUTE FUNCTION logging();
```

- Примените триггер к таблице Customer.

```
CREATE TRIGGER customer_logging
AFTER INSERT OR UPDATE OR DELETE ON Product
FOR EACH ROW EXECUTE FUNCTION logging();
```

- Примените триггер к таблице Employee.

```
CREATE TRIGGER employee_logging
AFTER INSERT OR UPDATE OR DELETE ON Product
FOR EACH ROW EXECUTE FUNCTION logging();
```

- Примените триггер к таблице Warehouse.

```
CREATE TRIGGER warehouse_logging
AFTER INSERT OR UPDATE OR DELETE ON Product
FOR EACH ROW EXECUTE FUNCTION logging();
```

- Примените триггер к таблице Supplier.

```
CREATE TRIGGER supplier_logging
AFTER INSERT OR UPDATE OR DELETE ON Product
FOR EACH ROW EXECUTE FUNCTION logging();
```

1.3 Проверка системы логирования

- Добавление новых данных в таблицу Product:

```
INSERT INTO product (product_category_name, price, warehouse_id)
VALUES ('Arabica Vip Pro', 200000, 1);
```

- Обновление данных в таблице Product:

```
UPDATE product
SET price = 95000
WHERE product_id = 4;
```

- Удаление данных из таблицы Product:

```
DELETE FROM product
WHERE product_id = 6;
```

- Проверка main_log

```
lab3=# select * from main_log;
```

log_id	operation_type	operation_date	user_operator	changed_data
1	I	2024-10-26 22:14:47.801626	postgres	({"product_id":6,"product_category_name":"Arabica Vip Pro","price":200000.00,"warehouse_id":1})
2	U	2024-10-26 22:16:57.623706	postgres	({"product_id":4,"product_category_name":"Typica","price":95000.00,"warehouse_id":4})
3	D	2024-10-26 22:17:26.847963	postgres	({"product_id":6,"product_category_name":"Arabica Vip Pro","price":200000.00,"warehouse_id":1})

2. Задачи по шифрованию данных.

2.1 Создание таблицы для хранения конфиденциальных данных

- Сначала нам нужно создать отдельную таблицу для хранения конфиденциальных данных, таких как токены или ключи доступа для каждого типа пользователей. Эта таблица должна быть отделена от основных таблиц базы данных.

```
CREATE TABLE secret_data (
    id SERIAL PRIMARY KEY,
    username TEXT,
    secret_token TEXT
);
```

2.2 Шифрование данных

- Мы будем использовать симметричный алгоритм шифрования для шифрования данных в таблице secret_data. Это гарантирует, что даже если данные будут утечены, их нельзя будет прочитать без ключа шифрования. Подготовка к использованию модуля pgcrypto: PostgreSQL имеет модуль pgcrypto, который поддерживает шифрование и расшифровку данных.

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

2.3 Создание ключа шифрования из пароля

- Шифровальный ключ будет создан из пароля суперпользователя с помощью хеш-функции (например, SHA-256). Это гарантирует, что ключ не хранится напрямую в базе данных.

```
lab3=# select encode(digest('Ch.u992mvd', 'sha256'), 'hex') as
encryption_key;
```

```
-----
ca69b9601669b11f98acf29d694ec0e6d52f581bef9ffbe01bf426a3c2e6418a
(1 row)
```

2.4 Шифрование данных

- При вставке данных в таблицу `secret_data` мы будем шифровать значение `secret_token` с помощью ключа, созданного на предыдущем этапе.

```
INSERT INTO secret_data (username, secret_token)
VALUES
('Chu',pgp_sym_encrypt('token_Ch','ca69b9601669b11f98acf29d694ec0e6d52f581bef9ffbe01bf426a3c2e6418a')),
('postgres',pgp_sym_encrypt('token_postgres','ca69b9601669b11f98acf29d694ec0e6d52f581bef9ffbe01bf426a3c2e6418a'));
```

2.5 Доказательство расшифровки данных

- Чтобы доказать, что даже администраторы не могут видеть данные в таблице без знания пароля, мы будем использовать метод расшифровки. Для расшифровки данных нам нужен исходный пароль. Без этого пароля администраторы не смогут расшифровать и прочитать содержимое таблицы `secret_data`

```
lab3=# select * from secret_data;
 id | username | secret_token
-----+-----
 1 | Chu      | \xc3d0d4070302cbda441b01d7b95274d23a0167245609f84edb96a5edbd2c8b5689019035e368e08cf678485dfc70a65a79480d94e9f8b21af40531c9ce1a89ca09596e9f547218265ef
 2 | postgres | \xc3d0d4070302571df6bfa22c5a2064d23f01406b15d947d495bc01aeb175c38236d0b2cb584474597a12109270332cc8686d8ada1e339139e6ee48af356ada4be450c8e76f019f4d9ce3a60ea7b153d5
(2 rows)
```

- Расшифровка данных:

```
SELECT username, pgp_sym_decrypt(secret_token::bytea,
'ca69b9601669b11f98acf29d694ec0e6d52f581bef9ffbe01bf426a3c2e6418a')
AS decrypted_token FROM secret_data;
username | decrypted_token
-----+-----
Chu      | token_Ch
postgres | token_postgres
(2 rows)
```

3. Задачи по разграничению доступа в БД:

3.1 Создание роли

- Сначала я создаю две разные роли для различных типов пользователей (не считая суперпользователя): роль `sales_role` (роль продавца) и роль `warehouse_role` (роль сотрудника склада).

-- Создание роли для продавца

```
CREATE ROLE sales_role NOLOGIN;
```

-- Создание роли для сотрудника склада

```
CREATE ROLE warehouse_role NOLOGIN;
```

3.2 Назначение привилегий

- Предоставление прав доступа к схеме:

```
GRANT USAGE ON SCHEMA coffee_shop_schema TO sales_role;
```



```
GRANT USAGE ON SCHEMA coffee_shop_schema TO warehouse_role;
```

- О правах:

- Сотрудник продаж (sales_role): Имеет доступ к просмотру таблиц, связанных с заказами и клиентами.
- Сотрудник склада (warehouse_role): Имеет доступ к просмотру и управлению таблицами продуктов и складов.

```
-- Предоставление прав доступа сотруднику продаж
```

```
GRANT SELECT ON Orders TO sales_role;
```

```
GRANT SELECT ON Customer TO sales_role;
```

```
-- Предоставление прав доступа сотруднику склада
```

```
GRANT SELECT, INSERT, UPDATE ON Product TO warehouse_role;
```

```
GRANT SELECT, INSERT ON Warehouse TO warehouse_role;
```

3.3 Доступ к таблицам логов

- Только суперпользователь имеет право просматривать таблицу логов, что помогает защищать конфиденциальную информацию. Вам необходимо убедиться, что никто, кроме суперпользователя, не может получить доступ к этой таблице логов.

```
-- Не предоставлять права другим ролям на просмотр таблицы логов
```

```
REVOKE ALL ON main_log FROM sales_role;
```

```
REVOKE ALL ON main_log FROM warehouse_role;
```

3.4 Проведение проверки прав доступа

- Вход с ролью sales_role:

```
lab3=# set role sales_role;
```

```
SET
```

```
lab3=> select * from orders;
```

```
order_id | order_date | total_amount | employee_id | customer_id
```

```
-----+-----+-----+-----+-----
```

1	2024-10-12	150000.00	2	1
2	2024-10-13	599999.00	2	2
3	2024-10-13	70000.00	3	3
4	2024-10-20	6699999.00	4	4

```
(4 rows)
```

```
lab3=> select * from product;
```

```
ERROR: permission denied for table product
```

- Вход с ролью warehouse_role:

```
lab3=> SET ROLE warehouse_role;
```

```
SET
```

```
lab3=> SELECT * FROM Product;
```

product_id	product_category_name	price	warehouse_id
1	Arabica	100000.00	1
2	Robusta	90000.00	2
3	Bourbon	96000.00	3
4	Typica	95000.00	4

```
(4 rows)
```

- Проверка доступа к таблице логов:

```
lab3=> SELECT * FROM Orders;
ERROR: permission denied for table orders
lab3=> set role sales_role;
SET
lab3=> SELECT * FROM main_log;
ERROR: permission denied for table main_log
lab3=> SET ROLE warehouse_role;
SET
lab3=> SELECT * FROM main_log;
ERROR: permission denied for table main_log
lab3=>
```

Вывод

В ходе выполнения лабораторной работы по разграничению доступа в базе данных были достигнуты следующие результаты:

1. **Создание ролей:** Были успешно созданы две роли — `sales_role` для сотрудников, занимающихся продажами, и `warehouse_role` для работников склада. Эти роли были определены в соответствии с потребностями различных классов пользователей информационной системы.
2. **Определение привилегий:** Для каждой роли были установлены четкие привилегии на доступ к таблицам базы данных. Принцип минимальных привилегий был применен, что означает, что каждой роли были предоставлены только необходимые права доступа к определенным таблицам, а доступ к другим таблицам был ограничен.
3. **Ограничения доступа к таблицам логирования:** Таблица логирования `main_log` была настроена так, что доступ к ней был предоставлен только суперпользователю, что обеспечило защиту от несанкционированного доступа к данным.
4. **Тестирование и демонстрация:** Были проведены тесты на доступ к данным для каждой роли. Проверки показали, что сотрудники, имеющие роль `sales_role`, смогли получить доступ только к таблицам, связанным с заказами и клиентами, в то время как сотрудники со статусом `warehouse_role` имели доступ к таблицам продуктов и склада. Попытки доступа к данным из таблицы логирования были успешно заблокированы для всех ролей, кроме суперпользователя.
5. **Выводы о безопасности:** Настройка разграничения доступа позволила создать надежную систему безопасности, предотвращающую несанкционированный доступ к данным. Внедрение ролей и управление привилегиями обеспечивает защиту конфиденциальной информации и соблюдение принципа минимальных привилегий.

Таким образом, лабораторная работа по разграничению доступа в базе данных была успешно выполнена, что продемонстрировало эффективность созданной системы безопасности.