**SQL injection (Внедрение SQL-кода** ). Автор: Хоанг Тхань Лич

# Что такое Внедрение SQL

**Внедрение SQL-кода** ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) *SQL injection* */ SQLi*)

— это метод, который позволяет злоумышленникам воспользоваться уязвимостью проверки ввода в веб-приложениях и сообщениями об ошибках, которые СУБД возвращает для внедрения, и выполнять недопустимые операторы SQL. SQL-инъекция может позволить злоумышленникам выполнять операции, удалять, вставлять, обновлять и т. д. в базе данных приложения, даже на сервере, на котором запущено приложение. SQL-инъекция широко известна как посредник атаки на веб-приложения, данные которых управляются системами управления базами данных, такими как SQL Server, MySQL, Oracle, DB2, Sysbase...

— один из распространённых способов взлома [сайтов](https://ru.wikipedia.org/wiki/Веб-сайт) и [программ](https://ru.wikipedia.org/wiki/Компьютерная_программа), работающих с [базами данных](https://ru.wikipedia.org/wiki/База_данных), основанный на внедрении в запрос произвольного [SQL](https://ru.wikipedia.org/wiki/SQL)-кода.

Внедрение SQL, в зависимости от типа используемой [СУБД](https://ru.wikipedia.org/wiki/Реляционная_СУБД) и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (*например, прочитать содержимое любых*[*таблиц*](https://ru.wikipedia.org/wiki/Отношение_(реляционная_модель))*, удалить, изменить или добавить*[*данные*](https://ru.wikipedia.org/wiki/Данные)), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

Разработчик прикладных программ, работающих с базами данных, должен знать о таких уязвимостях и принимать меры противодействия внедрению SQL.

# Распространенные типы ошибок

### *Не проверять escape-символы запроса*

Это тип ошибки SQL-инъекции, которая возникает, когда код, проверяющий входные данные в SQL-запросе, отсутствует. В результате конечный пользователь может сделать несколько неожиданных запросов к базе данных приложения. Следующая строка кода иллюстрирует эту ошибку:

statement = "SELECT \* FROM users WHERE name = '" + userName + "';"

Этот оператор предназначен для возврата определенных записей имени пользователя из таблицы пользователей. Однако, если переменная «userName» каким-то образом правильно введена злоумышленником, это может оказаться SQL-запросом с другой целью, чем предполагал автор приведенного выше кода. Например, мы вводим значение переменной userName следующим образом:

a' or 't'='t

Запрос можно понять следующим образом:

**SELECT** \* **FROM** users **WHERE** name = 'a' **or** 't'='t';

Если приведенный выше код используется в процедуре аутентификации, приведенный выше пример можно использовать для принудительного выбора действительного имени пользователя, поскольку 't'='t' всегда верно. В то время как большинство серверов SQL позволяют выполнять несколько запросов с помощью одного вызова, некоторые API-интерфейсы SQL, такие как mysql\_query в php, не допускают этого по соображениям безопасности. Это только предотвращает атаки хакеров с использованием отдельных операторов, но не мешает хакерам изменять слова в синтаксисе запроса. Значение переменной "userName" в приведенном ниже запросе приведет к удалению пользователей из таблицы пользователей, аналогично выбору всех данных из таблицы данных (по сути, открытие таблиц данных). информация каждого пользователя), этот пример иллюстрирует API, который позволяет выполнять несколько запросов одновременно:

a';DROP TABLE users; SELECT \* FROM data WHERE 't' = 't

Это приводит к окончательному синтаксису вышеуказанного запроса следующим образом:

**SELECT** \* **FROM** users **WHERE** name = 'a';**DROP** **TABLE** users; **SELECT** \* **FROM** **data** **WHERE** 't' = 't';

### *Ошибка безопасности внутри сервера базы данных*

Иногда уязвимости могут существовать в самом программном обеспечении сервера базы данных, как в случае с функцией mysql\_real\_escape\_string() серверов MySQL. Это позволит злоумышленнику выполнить успешную атаку SQL-инъекции на основе необычных символов Unicode, даже если ввод экранируется.

# Виды атак типа внедрения SQL-кода

Существует три основных класса атак, основанных на внедрении SQL-кода:

* Классическая SQL инъекция (Classic SQLi)
* SQL инъекция, основанная на эксплуатации выводимых СУБД сообщений об ошибках (Error-based SQLi)
* Слепая SQL инъекция (Blind SQLi)

# Принцип атаки внедрения SQL

Допустим, серверное [ПО](https://ru.wikipedia.org/wiki/Программное_обеспечение), получив входной параметр id, использует его для создания SQL-запроса. Рассмотрим следующий [PHP](https://ru.wikipedia.org/wiki/PHP)-скрипт:

$id = $\_REQUEST['id'];

$res = mysqli\_query("SELECT \* FROM news WHERE id\_news = " . $id);

Если на сервер передан параметр id, равный 5 (например так: *http://example.org/script.php?id=5*), то выполнится следующий [SQL](https://ru.wikipedia.org/wiki/SQL)-запрос:

**SELECT** \* **FROM** news **WHERE** id\_news = 5

Но если злоумышленник передаст в качестве параметра id строку -1 OR 1=1 (например, так: *http://example.org/script.php?id=-1+OR+1=1*), то выполнится запрос:

**SELECT** \* **FROM** news **WHERE** id\_news = -1 **OR** 1=1

Таким образом, изменение входных параметров путём добавления в них конструкций языка SQL вызывает изменение в логике выполнения SQL-запроса (в данном примере вместо новости с заданным идентификатором будут выбраны все имеющиеся в базе новости, поскольку выражение 1=1 всегда истинно — вычисления происходят [по кратчайшему контуру в схеме](https://ru.wikipedia.org/wiki/Вычисления_по_короткой_схеме)).

### ***Внедрение в строковые параметры***

Предположим, серверное ПО, получив запрос на поиск данных в новостях параметром search\_text, использует его в следующем SQL-запросе (здесь параметры экранируются кавычками):

$search\_text = $\_REQUEST['search\_text'];

$res = mysqli\_query("SELECT id\_news, news\_date, news\_caption, news\_text, news\_id\_author

FROM news WHERE news\_caption LIKE('%**$search\_text**%')");

Сделав запрос вида *http://example.org/script.php?search\_text=Test* мы получим выполнение следующего SQL-запроса:

**SELECT** id\_news, news\_date, news\_caption, news\_text, news\_id\_author **FROM** news

**WHERE** news\_caption **LIKE**('%Test%')

Но, внедрив в параметр search\_text символ кавычки (который используется в запросе), мы можем кардинально изменить поведение SQL-запроса. Например, передав в качестве параметра search\_text значение '**)+and+(news\_id\_author='1**, мы вызовем к выполнению запрос:

**SELECT** id\_news, news\_date, news\_caption, news\_text, news\_id\_author **FROM** news

**WHERE** news\_caption **LIKE**('%') **and** (news\_id\_author='1%')

### ***Использование UNION***

Язык SQL позволяет объединять результаты нескольких запросов при помощи оператора UNION. Это предоставляет злоумышленнику возможность получить несанкционированный доступ к данным.

Рассмотрим скрипт отображения новости (*идентификатор новости, которую необходимо отобразить, передается в параметре id*):

$res = mysqli\_query("SELECT id\_news, header, body, author FROM news WHERE id\_news = " . $\_REQUEST['id']);

Если злоумышленник передаст в качестве параметра id конструкцию **-1 UNION SELECT 4 username, password,1 FROM admin**, это вызовет выполнение SQL-запроса

**SELECT** id\_news, header, body, author **FROM** news **WHERE** id\_news = -1 **UNION** **SELECT** 1,username,password,1 **FROM** **admin**

Так как новости с идентификатором −1 заведомо не существует, из таблицы news не будет выбрано ни одной записи, однако в результат попадут записи, несанкционированно отобранные из таблицы admin в результате инъекции SQL.

### ***Использование UNION + group\_concat()***

В некоторых случаях хакер может провести атаку, но не может видеть более одной колонки. В случае MySQL взломщик может воспользоваться функцией:

group\_concat(col, symbol, col)

которая объединяет несколько колонок в одну. Например, для примера данного выше вызов функции будет таким:

-1 **UNION** **SELECT** group\_concat(username, 0x3a, password) **FROM** **admin**

### ***Экранирование хвоста запроса***

Зачастую SQL-запрос, подверженный данной уязвимости, имеет структуру, усложняющую или препятствующую использованию union. Например скрипт:

$res = mysqli\_query("SELECT author FROM news WHERE id=" . $\_REQUEST['id'] . " AND author LIKE ('a%')");

отображает имя автора новости по передаваемому идентификатору id только при условии, что имя начинается с буквы а, и внедрение кода с использованием оператора UNION затруднительно.

В таких случаях злоумышленниками используется метод экранирования части запроса при помощи символов комментария (**/\*** или **--** в зависимости от типа СУБД).

В данном примере злоумышленник может передать в скрипт параметр id со значением **-1 UNION SELECT password FROM admin/\***, выполнив таким образом запрос

**SELECT** author **FROM** news **WHERE** id=-1 **UNION** **SELECT** password **FROM** **admin***/\* AND author LIKE ('a%')*

в котором часть запроса (*AND author LIKE ('a%')*) помечена как комментарий и не влияет на выполнение.

### ***Расщепление SQL-запроса***

Для разделения команд в языке SQL используется символ **;** (*точка с запятой*), внедряя этот символ в запрос, злоумышленник получает возможность выполнить несколько команд в одном запросе, однако не все диалекты SQL поддерживают такую возможность.

Например, если в параметры скрипта

$id = $\_REQUEST['id'];

$res = mysqli\_query("SELECT \* FROM news WHERE id\_news = **$id**");

злоумышленником передается конструкция, содержащая точку с запятой, например **12;INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');** то в одном запросе будут выполнены 2 команды

**SELECT** \* **FROM** news **WHERE** id\_news = 12;

**INSERT** **INTO** **admin** (username, password) **VALUES** ('HaCkEr', 'foo');

и в таблицу admin будет несанкционированно добавлена запись HaCkEr.

# Методика атак типа внедрение SQL-кода

### Поиск скриптов, уязвимых для атаки

На данном этапе злоумышленник изучает поведение скриптов сервера при манипуляции входными параметрами с целью обнаружения их аномального поведения. Манипуляция происходит всеми возможными параметрами:

* Данными, передаваемыми через методы POST и GET
* Значениями [HTTP-Cookie]
* HTTP\_REFERER (для скриптов)
* AUTH\_USER и AUTH\_PASSWORD (при использовании аутентификации)

Как правило, манипуляция сводится к подстановке в параметры символа одинарной (реже двойной или обратной) кавычки.

Аномальным поведением считается любое поведение, при котором страницы, получаемые до и после подстановки кавычек, различаются (и при этом не выведена страница о неверном формате параметров).

Наиболее частые примеры аномального поведения:

* выводится сообщение о различных ошибках;
* при запросе данных (например, новости или списка продукции) запрашиваемые данные не выводятся вообще, хотя страница отображается

и т. д. Следует учитывать, что известны случаи, когда сообщения об ошибках, в силу специфики разметки страницы, не видны в браузере, хотя и присутствуют в её HTML-коде.

|  |  |  |  |
| --- | --- | --- | --- |
| **Конструкция** | **Комментирование остатка строки** | **Получение версии** | **Конкатенация строк** |
| **MySQL** | **--**..., **/\*** ..., или **#** ... | **version**() | **concat** (*string1*, *string2*) |
| **MS SQL** | **--** ... | **@@version** | *string1* + *string2* |
| **Oracle** | **--** ... или **/\*** ... | **select** banner **from** v$version | *string1* || *string2* или **concat** (*string1*, *string2*) |
| **MS Access** | Внедрение в запрос NULL‑байта: **%00**... |  |  |
| **PostgreSQL** | **--** ... | **SELECT version**() | *string1* || *string2*, CONCAT('a','b') |
| **Sybase** | **--** ... | **@@version** | *string1* + *string2* |
| **IBM DB2** | **--** ... | **select** versionnumber **from** sysibm.sysversions | *string1* || *string2* или *string1* **concat** *string2* |
| **Ingres** | **--** ... | **dbmsinfo**('\_version') | *string1* || *string2* |

# Защита от атак типа внедрение SQL-кода

Для защиты от данного типа атак необходимо тщательно фильтровать входные параметры, значения которых будут использованы для построения SQL-запроса.

### **Фильтрация строковых параметров**

Предположим, что код, генерирующий запрос (на языке программирования [Паскаль](https://ru.wikipedia.org/wiki/Паскаль_(язык_программирования))), выглядит так:

statement := 'SELECT \* FROM users WHERE name = "' + userName + '";';

Чтобы внедрение кода (закрытие строки, начинающейся с кавычки, другой кавычкой до её завершения текущей закрывающей кавычкой для разделения запроса на две части) было невозможно, для некоторых [СУБД](https://ru.wikipedia.org/wiki/Система_управления_базами_данных), в том числе, для [MySQL](https://ru.wikipedia.org/wiki/MySQL), требуется брать в [кавычки](https://ru.wikipedia.org/wiki/Кавычки) все строковые параметры. В само́м параметре заменяют кавычки на \", апостроф - на \', обратную косую черту - на \\ (это называется «[экранировать спецсимволы](https://ru.wikipedia.org/wiki/Экранирование_символов)»). Это можно делать таким кодом:

statement := 'SELECT \* FROM users WHERE name = ' + QuoteParam(userName) + ';';

**function** QuoteParam(s : **string**) : **string**;

*{ на входе — строка; на выходе — строка в кавычках и с заменёнными спецсимволами }*

**var**

i : integer;

Dest : **string**;

**begin**

Dest := '"';

**for** i:=1 **to** length(s) **do**

**case** s[i] **of**

'**''**' : Dest := Dest + '\**''**';

'"' : Dest := Dest + '\"';

'\' : Dest := Dest + '\\';

**else** Dest := Dest + s[i];

**end**;

QuoteParam := Dest + '"';

**end**;

Для PHP фильтрация может быть такой:

$query = "SELECT \* FROM users WHERE user='" . mysqli\_real\_escape\_string($user) . "';";

### **Фильтрация целочисленных параметров**

### **Усечение входных параметров**

### **Использование параметризованных запросов**

Возьмём другой запрос:

statement := 'SELECT \* FROM users WHERE id = ' + id + ';';

В данном случае поле **id** имеет числовой тип, и его чаще всего не берут в кавычки. Поэтому «закавычивание» и замена спецсимволов на escape-последовательности не проходит. В таком случае помогает проверка типа; если переменная **id** не является числом, запрос вообще не должен выполняться.

Например, на [Delphi](https://ru.wikipedia.org/wiki/Delphi_(язык_программирования)) для противодействия таким инъекциям помогает код:

**if** TryStrToInt(id, id\_int) **then**

statement := Format('SELECT \* FROM users WHERE id =%0:d;', [id\_int]);

Для PHP этот метод будет выглядеть так:

$query = 'SELECT \* FROM users WHERE id = ' . (int)$id;

Для внесения изменений в логику выполнения SQL-запроса требуется внедрение достаточно длинных строк. Так, минимальная длина внедряемой строки в вышеприведённых примерах составляет 8 символов («**1 OR 1=1**»). Если максимальная длина корректного значения параметра невелика, то одним из методов защиты может быть максимальное усечение значений входных параметров.

Например, если известно, что поле **id** в вышеприведённых примерах может принимать значения не более 9999, можно «отрезать лишние» символы, оставив не более четырёх:

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

Многие серверы баз данных поддерживают возможность отправки параметризованных запросов (подготовленные выражения). При этом параметры внешнего происхождения отправляются на сервер отдельно от самого запроса либо автоматически экранируются клиентской библиотекой. Для этого используются

* на [Delphi](https://ru.wikipedia.org/wiki/Delphi_(язык_программирования)) — свойство **TQuery.Params**;

Например

**var**

sql, param : **string**

**begin**

sql := 'select :text as value from dual';

param := 'alpha';

Query1.Sql.Text := sql;

Query1.ParamByName('text').AsString := param;

Query1.Open;

ShowMessage(Query1['value']);

**end**;

* на [Perl](https://ru.wikipedia.org/wiki/Perl) — через **DBI::quote** или **DBI::prepare**;
* на [Java](https://ru.wikipedia.org/wiki/Java) — через класс **PreparedStatement**;
* на [C#](https://ru.wikipedia.org/wiki/C_Sharp) — свойство **SqlCommand.Parameters**;
* на [PHP](https://ru.wikipedia.org/wiki/PHP) — [MySQLi](https://ru.wikipedia.org/wiki/MySQLi) (при работе с [MySQL](https://ru.wikipedia.org/wiki/MySQL)), **PDO**.