Доклад

Ошибки проверки вводимых данных: SQL-инъекция

Поляков Глеб Сергеевич

Содержание

1 Введение

SQL-инъекции представляют собой одну из наиболее распространенных и разрушительных атак на веб-приложения. Эта уязвимость позволяет злоумышленникам внедрять вредоносный SQL-код в запросы к базе данных через входные точки, такие как веб-формы, URL-параметры и другие пользовательские

входные данные. В результате таких атак злоумышленники могут получить несанкционированный доступ к данным, изменять или удалять их, что приводит к серьезным последствиям для бизнеса и пользователей.

SQL-инъекции остаются актуальной угрозой в современном мире информационных технологий. Согласно отчетам по безопасности, количество атак с использованием SQL-инъекций продолжает расти. За последние годы несколько крупных компаний стали жертвами подобных атак, что привело к утечкам персональных данных миллионов пользователей. Например, в 2017 году произошла известная утечка данных Equifax, затронувшая более 163 миллионов человек. Эта атака показала, насколько разрушительными могут быть последствия SQL-инъекций для организаций и их клиентов.

2 Основная часть

2.1 Принципы работы SQL-инъекции

SQL-инъекция работает за счет использования недостаточной обработки пользовательских данных перед их включением в SQL-запросы. Злоумышленник манипулирует вводимыми данными таким образом, чтобы изменять логику SQL-запросов, выполняемых на сервере базы данных. Ниже описаны основные принципы работы SQL-инъекции с примерами.

2.1.1 Принцип 1: Внедрение в строковые данные

Этот тип инъекции происходит, когда пользовательский ввод включается в строковые параметры SQL-запроса без должного экранирования или фильтрации.

2.1.1.1 Пример

Рассмотрим следующий SQL-запрос, уязвимый к инъекции:

```
query = "SELECT * FROM users WHERE username = '" + user_input + "'";
```

Если злоумышленник введет `admin' OR '1'='1`, результатом будет следующий SQL-запрос:

```
SELECT * FROM users WHERE username = 'admin' OR '1'='1';
```

Такой запрос вернет все записи из таблицы `users`, так как условие `1='1'` всегда истинно.

2.1.2 Принцип 2: Внедрение в числовые данные

Этот тип инъекции происходит, когда пользовательский ввод включается в числовые параметры SQL-запроса без должной проверки.

2.1.2.1 Пример

Рассмотрим следующий SQL-запрос:

```
query = "SELECT * FROM products WHERE product_id = " + product_id;
```

Если злоумышленник введет `0 OR 1=1`, результатом будет следующий SQL-запрос:

```
SELECT * FROM products WHERE product_id = 0 OR 1=1;
```

Такой запрос вернет все записи из таблицы `products`, так как условие `1=1` всегда истинно.

2.1.3 Принцип 3: Слепая SQL-инъекция (Blind SQL Injection)

Этот тип инъекции используется, когда сервер базы данных не возвращает результат запроса напрямую, но можно определить поведение приложения на основе истинности или ложности условий.

2.1.3.1 Пример

Рассмотрим следующий SQL-запрос:

```
query = "SELECT * FROM users WHERE username = '" + user_input + "' AND
password = '" + password + "'";
```

Злоумышленник может ввести `admin' AND '1'='1` в поле имени пользователя. Итоговый запрос будет выглядеть следующим образом:

```
SELECT * FROM users WHERE username = 'admin' AND '1'='1' AND password =
'password';
```

Если страница загружается успешно, злоумышленник может понять, что первая часть условия была выполнена.

2.1.4 Принцип 4: Инъекция на основе ошибок (Error-based SQL Injection)

Этот тип инъекции использует ошибки, возвращаемые сервером базы данных, для получения информации о структуре базы данных.

2.1.4.1 Пример

Рассмотрим следующий SQL-запрос:

```
query = "SELECT * FROM users WHERE username = '" + user_input + "'";
```

Если злоумышленник введет `admin' UNION SELECT null, table_name FROM information_schema.tables --`, результатом будет следующий SQL-запрос:

```
SELECT * FROM users WHERE username = 'admin' UNION SELECT null, table_name
FROM information_schema.tables --';
```

Этот запрос может вызвать ошибку или вернуть данные о таблицах базы данных, в зависимости от настроек сервера.

2.1.5 Принцип 5: Инъекция на основе времени (Time-based SQL Injection)

Этот тип инъекции используется, когда злоумышленник определяет истинность условий, добавляя задержки в выполнение SQL-запросов.

2.1.5.1 Пример

Рассмотрим следующий SQL-запрос:

```
query = "SELECT * FROM users WHERE username = '" + user_input + "'";
```

Злоумышленник может ввести `admin' OR IF(1=1, SLEEP(5), 0) --`, результатом чего будет следующий SQL-запрос:

```
SELECT * FROM users WHERE username = 'admin' OR IF(1=1, SLEEP(5), 0) --';
```

Если сервер задерживает ответ на 5 секунд, злоумышленник понимает, что условие было истинным.

Виды SQL-инъекций

##Виды SQL-инъекций

1. **Инъекция в строковых данных**: Происходит, когда инъекция внедряется в строковые параметры.
2. **Инъекция в числовых данных**: Злоумышленник вводит числовые значения, которые изменяют логику SQL-запроса.
3. **Слепая SQL-инъекция (Blind SQL Injection)**: Когда сервер не возвращает результат запроса напрямую, но поведение приложения изменяется на основе истинности или ложности условий.
4. **Инъекция на основе ошибок (Error-based SQL Injection)**: Злоумышленник использует ошибки, возвращаемые базой данных, для получения информации.

2.2 Методы защиты от SQL-инъекций

SQL-инъекции остаются одной из самых распространенных угроз безопасности для веб-приложений и баз данных. Чтобы защититься от таких атак, необходимо применять несколько методов, которые вместе создают многоуровневую систему защиты. Ниже описаны основные методы защиты от SQL-инъекций.

2.2.1 1. Использование подготовленных выражений (Prepared Statements)

Подготовленные выражения позволяют разделить SQL-запрос и данные, что предотвращает внедрение вредоносного кода. Они гарантируют, что введенные данные обрабатываются как параметры, а не как часть SQL-запроса.

2.2.1.1 Пример на Java:

```
String query = "SELECT * FROM users WHERE username = ?";
PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, user_input);
ResultSet rs = pstmt.executeQuery();
```

2.2.2 2. Хранимые процедуры (Stored Procedures)

Хранимые процедуры выполняются на сервере базы данных и позволяют выполнять предопределенные операции. Они помогают избежать SQL-инъекций, так как параметры процедуры обрабатываются отдельно от тела запроса.

2.2.2.1 Пример на SQL Server:

```
CREATE PROCEDURE GetUserByUsername
    @username NVARCHAR(50)
AS
BEGIN
    SELECT * FROM users WHERE username = @username;
END;
```

Вызов хранимой процедуры:

```
EXEC GetUserByUsername @username = 'user_input';
```

2.2.3 3. Экранирование специальных символов

Экранирование специальных символов предотвращает их интерпретацию как части SQL-запроса. Это может быть полезно для баз данных и приложений, которые не поддерживают подготовленные выражения или хранимые процедуры.

2.2.3.1 Пример на PHP:

```
$username = mysqli_real_escape_string($connection, $user_input);
$query = "SELECT * FROM users WHERE username = '$username'";
$result = mysqli_query($connection, $query);
```

2.2.4 4. Валидация и фильтрация данных

Валидация и фильтрация данных помогает убедиться, что вводимые пользователем данные соответствуют ожидаемому формату и содержат только допустимые значения.

2.2.4.1 Пример на PHP:

```
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    // valid email
} else {
    // invalid email
}
```

2.2.5 5. Принцип минимальных привилегий

Принцип минимальных привилегий означает, что учетная запись базы данных, используемая веб-приложением, должна иметь только те права, которые необходимы для выполнения конкретных задач. Например, для операций чтения данных не нужны права на изменение или удаление данных.

2.2.5.1 Пример:

Создайте пользователя базы данных с минимальными привилегиями:

```
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE ON database_name.* TO 'app_user'@'localhost';
```

2.2.6 6. Использование ORM (Object-Relational Mapping)

ORM-системы абстрагируют работу с базой данных и автоматически обрабатывают параметры, что снижает риск SQL-инъекций.

2.2.6.1 Пример на Python с использованием SQLAlchemy:

```
from sqlalchemy import create_engine, Table, MetaData
engine = create_engine('mysql+pymysql://user:password@localhost/database')
metadata = MetaData(bind=engine)
users = Table('users', metadata, autoload=True)
stmt = users.select().where(users.c.username == user_input)
result = engine.execute(stmt)
```

2.2.7 7. Обнаружение и предотвращение вторжений (IDS/IPS)

Системы обнаружения и предотвращения вторжений могут мониторить трафик и обнаруживать подозрительные активности, такие как попытки SQL-инъекций. Они могут блокировать или предупреждать администратора о возможных атаках.

2.2.8 8. Регулярное тестирование безопасности

Проведение регулярных проверок безопасности и тестов на проникновение помогает выявить и устранить уязвимости до того, как их смогут использовать злоумышленники. Важно применять инструменты для автоматического сканирования и проводить ручные проверки.

Таблица 1: методы защиты от SQL-инъекций

Метод защиты	Описание	Пример
Подготовленные выражения	Разделяют SQL-запрос и данные, предотвращая внедрение вредоносного кода.	`PreparedStatement pstmt = connection.prepareStatement("SELECT * FROM users WHERE username = ?");`
Хранимые процедуры	Выполняют предопределенные операции на сервере базы	`CREATE PROCEDURE GetUserByUsername @username`

Метод защиты	Описание	Пример
	данных, исключая изменение структуры запроса злоумышленником.	`NVARCHAR(50) AS BEGIN SELECT * FROM users WHERE username = @username; END;`
Экранирование специальных символов	Обработка специальных символов, чтобы предотвратить их интерпретацию как части SQL-запроса.	`mysqli_real_escape_string($connection, $user_input);`
Валидация и фильтрация данных	Проверка ввода данных на соответствие ожидаемому формату и допустимым значениям.	`if (filter_var($email, FILTER_VALIDATE_EMAIL)) { // valid email } else { // invalid email }`
Принцип минимальных привилегий	Учетная запись базы данных должна иметь только необходимые права доступа.	`GRANT SELECT, INSERT, UPDATE ON database_name.* TO 'app_user'@'localhost';`
Использование ORM	Абстрагирование работы с базой данных, автоматическая обработка параметров.	`stmt = users.select().where(users.c.username == user_input);`
IDS/IPS	Системы обнаружения и предотвращения вторжений для мониторинга и блокировки подозрительных активностей.	Использование специализированных IDS/IPS решений.
Регулярное тестирование безопасности	Проведение проверок безопасности и тестов на проникновение для выявления и устранения уязвимостей.	Использование инструментов для автоматического сканирования и проведения ручных проверок.

2.3 Заключение

SQL-инъекции представляют серьезную угрозу безопасности данных и приложений. Понимание принципов работы SQL-инъекций позволяет лучше защищаться от них. Важные меры защиты включают использование подготовленных выражений, хранимых процедур, экранирование специальных символов, валидацию данных и ограничение прав доступа. Внедрение этих мер значительно снижает риск успешной SQL-инъекции.

Здесь описываются теоретические аспекты, связанные с выполнением работы.

Например, в табл. 1 приведено краткое описание стандартных каталогов Unix.

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 1).

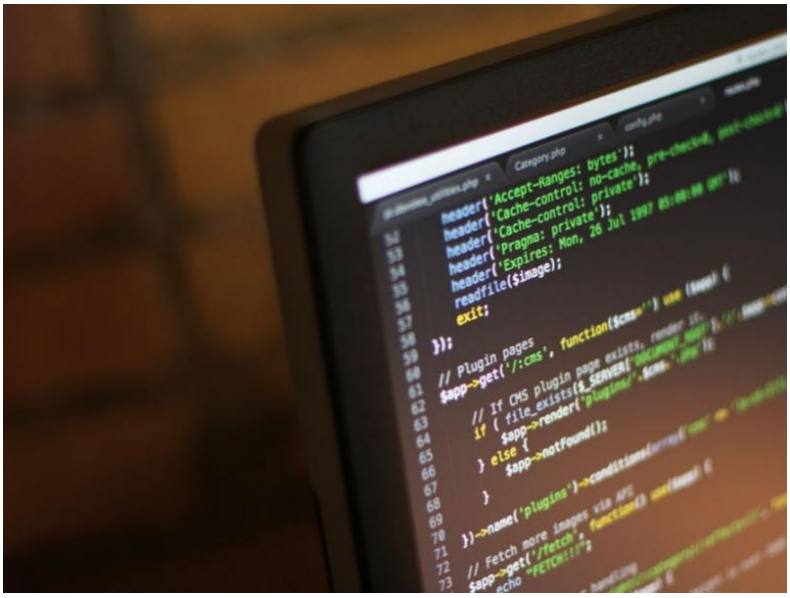

Рис. 1: Название рисунка

3 Выводы

SQL-инъекции остаются актуальной и серьезной угрозой для информационных систем. Однако, правильное понимание принципов работы SQL-инъекций и внедрение эффективных методов защиты позволяют значительно снизить риск таких атак. Применение подготовленных выражений, хранимых процедур, экранирование данных, валидация ввода, принцип минимальных привилегий, использование ORM, IDS/IPS-системы и регулярное тестирование безопасности составляют основу комплексной стратегии защиты. Следуя этим принципам, организации могут обеспечить высокую степень безопасности своих данных и приложений, защищая их от потенциальных атак SQL-инъекций.

Список литературы