万字sql注入学习过程--------从一个查询语句到SQL注入原理——包含各种注入姿势及相关CTF，靶场，漏洞实战 随时补充内容

原创

置顶 AAAAAAAAAAAA66 已于 2022-02-04 00:06:40 修改 1935 收藏 3
分类专栏： 漏洞原理解析 CTF -WEB 学习 文章标签： php apache 网络安全
于 2021-11-30 00:37:09 首次发布
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/AAAAAAAAAAAA66/article/details/121622115
版权


漏洞原理解析 同时被 2 个专栏收录
12 篇文章 0 订阅
订阅专栏


CTF -WEB 学习
34 篇文章 1 订阅
订阅专栏

最近一次更新：2022/2/3 更新内容：DNSLOG注入，SOAP注入，及相关实战

从第一次接触到SQL注入到现在已经有小半年了，简单打了个DVWA靶场之后便没怎么用手工注入，非常依赖工具，最近遇到几道需要手动注入的CTF题目，发现自己对sql注入的原理只是停留在一个基础的理论之上，所以就写这一篇文章对自己的sql注入进行新的梳理。当然sql注入涉及的内容非常多，甚至涉及一些底层原理，想要学好还是得不断动手，不断思考源码。

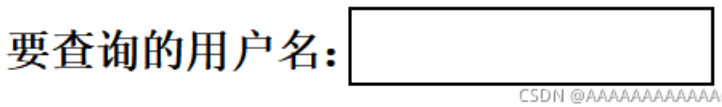
> 本文的目的也只是尽量的全面介绍，适合作为入门学习，通过一些做实例不断的思考，有那个点不懂就去搜那个知识点，这也是我学习的过程。

目录

CSDN @AAAAAAAAAAAA66

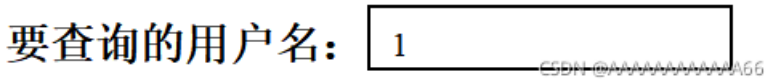
开始介绍sql注入漏洞之前，先介绍一个基础的mysql查询语句（学过数据库的同学直接跳走）

我们经常遇到一个这样的需要我们输入的表格



CSDN @AAAAAAAAAAAA66

我们在这输入一个 1



那么在数据库中就会自动执行这样的语句

select * from users where id = 1

select * from xxxx where xxx=AAA 这个是数据库的一般查询语句 的格式

SQL注入简介

sql注入就是指web应用程序对用户输入数据和合法性没有判断，前端传入后端的参数是攻击者可控的，并且参数带入数据库查询，攻击者可以通过构造不同的sql语句来实现对数据库的任意操作。

SQL注入产生的条件

sql注入漏洞的产生需满足以下两个条件。

- 参数用户可控：前端传给后端的参数内容是用户可以控制的
- 参数带入数据库查询：传入的参数拼接到sql语句且带入数据库查询。

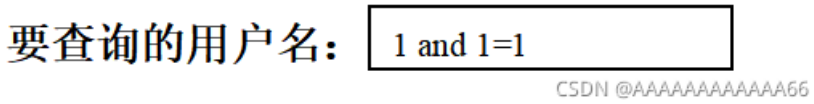
（输入的参数会被数据库执行，也就是可以编写恶意代码让数据库执行）

SQL注入漏洞的类型及判断存在的方法

数字型注入

数字型一般的查询格式为：
select * from user where id =x (其中x是我们能选择输入的参数)

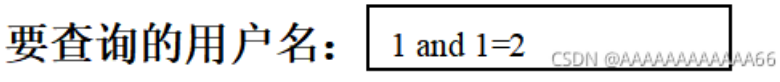
这个时候我们可以使用万能语句 1 and 1=1



那么数据库就会执行

```
select * from user where id=1 and 1=1
```

(出现正常回显)



同理数据库执行 （这个语句永远不成立，所以会出现异常）

```
select * from user where id=1 and 1=2
```

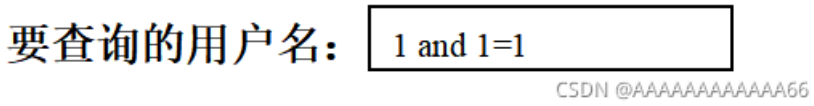
(不回显，或者报错)

通过以上这2步我们可以确认存在数据库满足2个条件，执行了我们输入的异常参数，所以可以得出结论，存在数字型注入。

字符型注入

字符型注入一般的查询格式为：
select * from user where id ='x'　　(可以很直观的看到，相比数字型注入，查询语句自动给我们加上了一对双引号")

如果这个时候我们用数字型注入的方式 提交数据为 1 and 1=1



CSDN @AAAAAAAAAAAA66

那么到数据库中执行的语句为

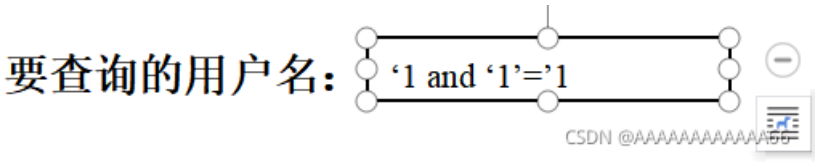
```
select * from user where id='1 and 1=1'
```

很显然，这个语句不成立。

同理输入 1 and 1=2，也不成立，这时候我们就无法判断数据库是否执行了我们的语句。

所以对于字符型注入的判断，我们采取的是闭合双引号的方法。

我们输入



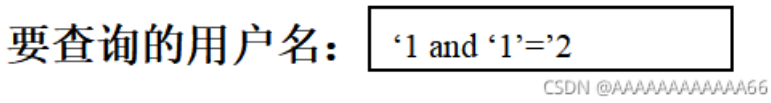
CSDN @AAAAAAAAAAAA66

那么到数据库查询的语句为：

```
select * from user where id='1' and '1'='1'
```

这个时候，如果数据库正常处理了我们输入的参数，显然是会正常输出原来的界面的。

我们再输入



CSDN @AAAAAAAAAAAA66

数据库查询

```
select * from user where id='1' and '1'='2'
```

如果这个时候报错，我们就能判断数据库存在字符型注入。

Mysql数据库的相关知识（给有数据库基础的同学）

由于我们一般实验环境是在mysql数据库的，所以这里对mysql数据库一些需要注意的地方进行说明。需要指出的是SQL注入是一个大类（写完全要一本很厚的书），设计到很多种不同的数据库，这里的讲解只是为了更好的理解SQL注入的原理。

mysql 5.0以下

mysql 5.0以下，多用户单操作。因为现在基本上很少用mysql5.0以下的版本了，这里就不过多叙述。

mysql 5.0 以上（重点）

多用户多操作。

> 在mysql 5.0版本之后，Mysql默认在数据库中存放一个 “information_schema”的数据库，在该库中，我们需要记住三个表名，分别是SCHEMATA,TABLES和COLUMNS.(数据库中不区分大小写，所以下面我用小写，其实都是一样的)

- schemata表存储该用户创建的所有数据库的库名，我们需要记住该表中记录数据库库名的字段名为schema_name
- tables表存储该用户创建的所有数据库的库名和表名，我们要记住该表中记录数据库库名和表名的字段名分别为table_schema和table_name
- columns表存储该用户创建的所有数据库的库名，表名，和字段名，我们需要记录该表中记录数据库库名，表名和字段名为table_schema,table_name和column_name.

看了这么多，说人话就是mysql自带一个数据库，把用户自己创建的数据库信息全部记录了下来，所以，搞定了information_schema数据库，就搞定了这个用户全部的数据库，和储存的信息。

注入实例（ctf题目）

下面是一些sql注入的实例，大家可以看一下顺便理清原理和思路，光讲原理的话非常空洞（这个会不断更新的，我做过的注入题都会放在这里面，也推荐大家到相关网站去练习，边练边学边思考是最好的）

i 春秋 CTF训练营 web——SQL注入 1（字符型注入）（手动注入）

一鱼三吃 i春秋CTF-训练营 SQL注入-2 sqlmap bp手注 python脚本 （post提交表单 字符型注入）

春秋 “百度杯”CTF比赛 九月场 SQL %00截断绕过 （简单的绕过）

BMZCTF 强网杯 2019 随便注 原理+题解_AAAAAAAAAAA66的博客-CSDN博客
buuctf XCTF October 2019 Twice SQL Injection 二次注入原理+题解_AAAAAAAAAAA66的博客-CSDN博客

二次注入攻击_糖小喵-CSDN博客_二次注入攻击

安鸾靶场X-Forwarded-For注入头注入，Cooike注入训练，手注+sqlmap操作

特别的sql注入方式-DNSlog注入--安鸾靶场练习 DNSlog注入学习_AAAAAAAAAAA66的博客-CSDN博客

SOAP注入学习——安鸾靶场---SOAP协议注入 练习记录_AAAAAAAAAAA66的博客-CSDN博客

注入语句类型

mysql联合查询注入（常用）

> mysql联合查询注入利用UNION（联合查询）可以同时执行多条SQL语句的特点，在参数中插入恶意的SQL注入语句，同时执行两条SQL语句，获取额外敏感信息或者执行其他数据库操作。
>
> 介绍一些语句（假设我们通过判断得出网站存在数字型注入）
>
> 通过 id=1 order by 1~99 判断得出数据库的列数为3

（1）判断报错点（可注入点）

```
?id=1 union select 1,2,3
```

（2）获取数据库名

```
?id=1 union select(1,2,database())
```

（3）获取表名

```
?id=1 union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='xxx'
```

（4）获取字段名

```
?id=1 union select 1,group_concat(column_name),3 from information_schema.columns where table_name='xxx'
```

（5）获取字段值

```
?id=1 union select 1,xxx,3 from xxxx
```

xxx为字段名　xxxx为表名

这里主要是利用UNION语句，我们可以先看这道题，有更细致的步骤。

结合这个运用能理解的更快。（我们不需要一步到位的全部理解每个语句，每个字符串的实际意义，只需要看流程就行，那个知识点不懂 比如说order by 的原理，可以自行搜索）

i 春秋 CTF训练营 web——SQL注入 1（字符型注入）（手动注入）

报错注入（常用）

> 这种注入形式一般存在在输入错误的数据时，比如?id=1"""""(多个引号导致报错)，然后程序会将错误的信息输出，一般我们用updatexml（）函数。

（1）获取管理员的用户名。

```
'and updatexml(1,concat(0x7e,(select user()),0x7e),1)--+
```

（2）获取数据库名

```
'and updatexml(1,concat(0x7e,(database()),0x7e),1)--+
```

（3）获取数据库中数据库名（这句话没有错，不懂得可以看上面得mysql5.0 information_schema

mysql 5.0 以上都会自带这个数据库，information_schema包含mysql数据库中所有的数据库名，及其表名，字段名）

```
'and updatexml(1,concat(0x7e,(select schema_name from information_schema.schemata limit 0,1  ),0x7e),1)--+
```

（报错注入只显示一条结果，所以要用limit语句）

（4）查询表名

```
'and updatexml(1,concat(0x7e,(select table_name from information_schema.tables where table_schema='xxxxx' l
```

（5）查询字段名(xxx为表名)

```
'and updatexml(1,concat(0x7e,(select column_name from information_schema.columns where table_schema='xxx' l
```

（6）查询字段值

```
'and updatexml(1,concat(0x7e,(select xxx from xxxx limit 0,1),0x7e),1)--+
```

xxx为（字段名）xxxx为（表名）

给道例题

报错注入例题

Boolean注入（常用）

> MYSQL bool注入是盲注的一种，与报错注入不同，bool注入没有任何报错信息输出，页面返回只有正常和不正常两种状态，攻击者只能通过返回的这两个状态来判断输入的SQL注入测试语句是否正确，从而判断数据库中存储了那些信息。(但是效率较低，一般配合Python脚本使用)

（1）获取数据库长度(判断数据库长度是否大于8)不断测试

```
?id=1 and (select length (database()))>8
```

（2）获取当前数据库名第一个字符的ASCLL码是否大于108 （在第一题找到数据库的长度的前提下，不断测试出每一个字符）

```
?id=1 and (select ascii (substring (database(),2,1))>108
```

（3）。。明白个意思就行，实际上不太可能手注，一般就配合脚本。

之后会更新个sql注入脚本的

Mysql sleep注入

> sleep 注入是另一种形式的盲注，与bool注入不同，sleep注入没有任何报错信息输出，页面不返回不管对或者错都是一种状态，攻击者无法通过页面返回状态来判断输入的SQL注入测试语句是否正确，只能通过构造sleep注入的SQL测试语句，根据页面的返回时间判断数据库中是否存储了那些信息。（说人话就是前面的招数用了都没回显，被逼无奈只能用这个方法，更加耗费时间）

（1）判断数据库的长度

```
?id=1 and sleep (if(length((select database()))=10 0,5 )
```

等于10的话就立马回显，不等于就过5秒返回。

（2）判断数据库的名称，例如判断数据库的第一个字符：

```
?id=1 and sleep (if (ascii(substring(database(),1,1))<116,0,5))
```

明白了原理 下面步骤就不一一列出了，搞懂了前面的自然后面的也懂，

floor注入

> floor注入是报错注入的一种方式，主要原因是rand函数与group by 字句一起使用时，rand函数1会计算多次，会导致报错产生的注入。下面给出语句示例

获取数据库名称

?id=1 and (select 1 from (select count (*),concat(database(),floor(rand(0)*2)) x from information_schema.tables group by x)a)

获取表名

?id=1 and (select 1 from (select count (*),concat((select(table_name)from information_schema.tables where table_schema=database() limit 0,1),floor(rand(0)*2)) x from information_schema.tables group by x)a)

获取列名

?id=1 and (select 1 from (select count (*),concat((select(colunmn_name)from information_schema.columns where table_schema=database() and table_name='xxx' limit 0,1),floor(rand(0)*2)) x from information_schema.tables group by x)a)

获取数据

?id=1 and(select 1 from (select count (*),concat ((select username from xxx limit 0,1),0x3a,floor(rand()*2))x from information_schema.tables group by x)a)

原理

floor注入本质上还是属于报错注入。是由ran函数在与group by 字句一起用时多次计算而导致的。

- floor函数：floor(x)返回不大于x的最大整数值
- rand函数：返回一个0~1的随机数

另外还要知道MYSQL在执行select count(*) from tables group by x 这类语句时会创建一个虚拟表，然后在虚拟表中插入数据。在执行floor(rand())函数时会出现报错。

篇幅太长了， 具体实现过程可以看下面的链接，

MYSQL floor 报错注入详解_AaronLuo-CSDN博客_floor报错注入原理

宽字节注入

> 宽字节注入的产生：开发者为了防止出现SQL注入攻击，将用户输入的数据用addslashes等函数进行过滤。addslashes等函数默认对单引号等字符进行转义，这样就可以避免注入。
>
> 而MYSQL在使用GBK编码的时候，如果第一个字符的ASCLL码大于128，会认为前两个字符是一个汉字，会将后面的转义字符\,"吃掉"，将前2个字符凭借成汉字，这样就可以将SQL语句闭合，造成宽字节注入。
>
> 说人话就是：开发者原本想转义单引号，但是在MYSQL使用GBK编码的时候，我们可以通过一些改动使得他的过滤失效，仍然能闭合单引号，从而导致sql注入。

查看数据库名称（下面不一定非要%81，大于%80就行，这样就在汉字的编码）另外一个%23 #是为了注释掉最后面的单引号

```
?id=1%81' and 1=2 union select 1,database(),3 %23
```

这里比较一下

如果我们输入

```
?id=1' and 1=2 union select 1,database(),3 %23
```

这里服务器将单引号转义，数据库执行就会报错，得不到结果。

然后我们又会面对下一个问题，我们在接下来的查询中，还是会需要用到单引号，

?id=1%81' and 1=2 union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='xxx' %23

这样们知道数据库为xxx，但这里还是要遇到单引号，那这样要怎么办呢？

下面有2种方法

1. 16进制转化，MYSQL是可以正常处理16进制的，所以可以用table_schema=0x787878代替table_schema='xxx'
2. 嵌套查询，table_name=(select database()),这样就替换了table_schema='xxx'

下面用嵌套查询查询字段值，（查询的是第一个表名，如果还想查询其他表名，自行修改limit后面的数字）

select column_name from information_schema.columns where table_schema=(select database()) and table_name(select table_name from information_schema.tables where table_schema=(select datbase())limit 0,1)limit 0,1

```
select column_name from information_schema.columns where table_schema=(select database()) and table_name(se
```

这里使用三层嵌套，没见过的属实劝退。。。。

不过不会也没关系，可以用16进制转化。

堆叠注入

> 堆叠查询注入：堆叠查询可以执行多条SQL语句，语句之间以分号(;)隔开。而堆叠查询注入攻击就是利用此特点，在第二条语句中构造自己要执行的语句。

比如：　　(这样一下就会执行2条命令，与union注入有相似的地方，union注入可以在一条语句中执行多个操作，而堆叠注入可以在一次输入中执行多条语句)

```
?id=1 ;show database;select xxx from xxx; 一次执行三语句
```

　　虽然这个注入姿势很牛逼，但实际遇到很少，其可能受到API或者数据库引擎，又或者权限的限制只有当调用数据库函数支持执行多条sql语句时才能够使用，利用mysqli_multi_query()函数就支持多条sql语句同时执行，但实际情况中，如PHP为了防止sql注入机制，往往使用调用数据库的函数是mysqli_ query()函数，其只能执行一条语句，分号后面的内容将不会被执行，所以可以说堆叠注入的使用条件十分有限，一旦能够被使用，将可能对网站造成十分大的威胁。

细节原理：SQL注入之堆叠注入_沫忆末忆的博客-CSDN博客_堆叠注入

实战题目：BMZCTF 强网杯 2019 随便注 原理+题解_AAAAAAAAAAA66的博客-CSDN博客

二次注入

二次注入的成因

- **由于开发者为了防御sql注入，使网站对我们输入的恶意语句中的一些重要的关键字进行了转义，使恶意的sql注入无法执行（比如说将单引号转义，使其无法闭合）。**
- **但是数据库存储我们的数据时，输入的恶意的语句又被还原成转义之前的语句（数据库又没有对存储的数据进行检查，默认存储的数据都是无害的）**
- **这时仍然没有被攻击，但是当我们数据库在进行查询时，如果调用了这条信息，就可能会产生sql注入。**

所以需要说明的是，二次注入的产生是需要以上一些特定的条件的。所以二次注入一般比较难发现。

二次注入出现的场景

- **用户注册和登陆 通过登陆注册操作爆数据库信息**
- **用户修改密码　通过二次注入的特性修改别的用户的密码**

实例：登陆和注册

buuctf XCTF October 2019 Twice SQL Injection 二次注入原理+题解_AAAAAAAAAAA66的博客-CSDN博客

实例：修改密码

二次注入攻击_糖小喵-CSDN博客_二次注入攻击

Cookie注入

cooike简介：

一、什么是cookie

1.cookie是服务器通知浏览器保存一种键值对数据的一种技术

2.cookie由服务端产生，发生并通知浏览器保持cookie。

3.浏览器有了cookie以后，每次请求都会把cookie带给服务器。

说人话：cooike就相当于给你制作了个二维码，服务器通过检测这个二维码判断是不是本人。

cooike注入：服务器后台获取我们发送的cooike，然后直接将其放入到数据库中进行查询。

如果我们修改cooike，且后台没有在放入数据库查询之前进行过滤或者预编译，就会产生注入。

其实学到这，注入的原理都是一样的。无非只是出现的位置不同。

实例

安鸾靶场X-Forwarded-For注入头注入，Cooike注入训练，手注+sqlmap操作_AAAAAAAAAAA66的博客-CSDN博客

XFF注入

X-Forwarded-For 是一个 HTTP 扩展头部，主要是为了让 Web 服务器获取访问用户的真实 IP 地址，但是这个IP却未必是真实的。

同样后台会对接收的XFF头放在数据库进行查询，如果没有对输入的数据进行检测的话，还是会造成sql注入。

实例：安鸾靶场X-Forwarded-For注入头注入，Cooike注入训练，手注+sqlmap操作_AAAAAAAAAAA66的博客-CSDN博客

BASE64注入

base64简介：“Base64是网络上最常见的用于传输8Bit字节码的编码方式之一，Base64就是一种基于64个可打印字符来表示二进制数据的方法”。

这个其实遇见的很少，至少我没遇到过，不过也很简单，服务器会将我们输入的数据进行base64解码，然后在数据库中进行查询。
实际注入时，只要将自己的注入语句 base64编码就行。

不过这样？似乎是不必要将它单独划分为一个注入方式吧？

实际上，还有另外一种场景，加入网站有WAF（通过匹配正则），常用的sql注入语句全被过滤。
假如有这样一个功能模块，接受你的base64编码数据，在后台解码放入数据库查询。
如果你在这个地方输入sql注入语句，WAF就有可能无法拦截，（因为如果waf在后台的话，由于往后台传输的数据经过了base 64 编码 ，waf就有可能识别不出来）

dnslog注入（较为特殊）

特别的sql注入方式-DNSlog注入--安鸾靶场练习 DNSlog注入学习_AAAAAAAAAAA66的博客-CSDN博客

SOAP注入（较为特殊）

SOAP注入学习——安鸾靶场---SOAP协议注入 练习记录_AAAAAAAAAAA66的博客-CSDN博客

sql注入的防御以及自己的思考

> 成因：SQL 注入漏洞存在的原因，就是拼接 SQL 参数。也就是将用于输入的查询参数，直接拼接在 SQL 语句中，导致了SQL 注入漏洞。
>
> 那么如何防御呢？说人话有2种方法
>
> - 正则式匹配过滤危险字符
> - 使用预编译语句（主流，简单而方便）
>
> 2种方法的原理有什么不同呢？我们从sql注入的成因来思考：
>
> 我们输入 select union and ‘‘ -- = + # 这些我们注入时常用的字符，而如果服务器对我们输入的字符进行过滤，比如以前常见的过滤 " （双引号）判断存在就立即返回。比如说我们经常注册用户名时 不能有@ ’‘ % ……& 这一类的字符，都是这个道理，当然它们也不是都为了全为了防范sql注入（危险字符和很多漏洞有关，）
>
> （跑题了）。。 那么我们将 " select union 等语句过滤了，是否就安全了呢？不是的，还是能得（大小写，双写，编码绕过等。即使考虑到这些种种，其实还是有绕过的可能性的。
>
> 使用预编译语句：
>
> ```
> String sql = "select id, no from user where id=?";
> PreparedStatement ps = conn.prepareStatement(sql);
> ps.setInt(1, id);
> ps.executeQuery();
> ```
>
> 如上所示，就是典型的采用 sql语句预编译和绑定变量 。为什么这样就可以防止sql 注入呢？
>
> 其原因就是：采用了PreparedStatement，就会将sql语句："select id, no from user where id=?" 预先编译好，也就是SQL引擎会预先进行语法分析，产生语法树，生成执行计划，也就是说，后面你输入的参数，无论你输入的是什么，都不会影响该sql语句的 语法结构了，因为语法分析已经完成了，而语法分析主要是分析sql命令，比如 select ,from ,where ,and, or ,order by 等等。所以即使你后面输入了这些sql命令，也不会被当成sql命令来执行了，因为这些sql命令的执行， 必须先的通过语法分析，生成执行计划，既然语法分析已经完成，已经预编译过了，那么后面输入的参数，是绝对不可能作为sql命令来执行的，**只会被当做字符串字面值参数**。所以sql语句预编译可以防御sql注入。
>
> 具体原理可看下方链接。

SQL 注入防御方法总结_阳光灿烂的日子的博客-CSDN博客_sql注入防御的五种方法

作者水平有限，不当之处欢迎指出。