几期『三个白帽』小竞赛的writeup

转载

weixin_34004576 于 2018-03-08 11:02:40 发布 155 收藏
文章标签： php 数据库 运维
原文链接：https://juejin.im/post/5aa11850f265da238a3005cc

> phith0n · 2015/11/19 10:26

自从三个白帽问世以后，收到了大家的喜欢，依托『三个白帽』乌云做了几次小竞赛，我也出了几道题。Writeup不全是大家普遍反映的问题，我这里把几道题的解题思路汇总一下。 这几道题的源代码与环境都在三个白帽的集市中，大家获取三个白帽的邀请码以后可以在集市中进行购买与启动。

0x00 二次注入+文件名修改导致getshell

本题是出现在XDCTF2015线下决赛中的题目之一，被我移植到三个白帽的环境中了。考察的是代码审计功底，和对于二次注入的利用。

入口：二次注入漏洞

此题入口点是二次注入。
在common.inc.php中可以看到全局进行了转义，这样常规注入少了大部分。遍观代码，输入处没有任何反转义、反解压、数字型等特殊情况，基本可以确定不存在直接的注入漏洞。 看到上传处的代码upload.php：

```
#!php
$name = basename($file["name"]);
$path_parts = pathinfo($name);

if(!in_array($path_parts["extension"], ["gif", "jpg", "png", "zip", "txt"])) {
    exit("error extension");
}
$path_parts["extension"] = "." . $path_parts["extension"];

$name = $path_parts["filename"] . $path_parts["extension"];
$path_parts["filename"] = $db->quote($path_parts["filename"]);

$fetch = $db->query("select * from `file` where
`filename`={$path_parts['filename']}
and `extension`={$path_parts['extension']}");
if($fetch && $fetch->fetchAll()) {
    exit("file is exists");
}

if(move_uploaded_file($file["tmp_name"], UPLOAD_DIR . $name)) {
    $re = $db->exec("insert into `file`
          ( `filename`, `view`, `extension`) values
          ( {$path_parts['filename']}, 0, '{$path_parts['extension']}')");
    if(!$re) {
        print_r($db->errorInfo());
        exit;
    }
复制代码
```

可见，上传的文件名走过的流程是：
`$file['name'] -> pathinfo() -> $path_parts["filename"] -> quote() -> insert`
由于经过了pdo的quote方法转义，所以此处也不存在注入。
再看到rename.php

```
#!php
$result = $db->query("select * from `file` where `filename`='{$req['oldname']}'");
if ($result) {
    $result = $result->fetch();
}


if(!$result) {
    exit("old file doesn't exists!");
} else {

    $req['newname'] = basename($req['newname']);
    $re = $db->exec("update `file` set
                `filename`='{$req['newname']}',
                `oldname`='{$result['filename']}'
                where `fid`={$result['fid']}");
复制代码
```

根据`$req['filename']`从数据库里查询到已存在的一行，并调用update语句进行修改。
但这里`oldname='{$result['filename']}'`将从数据库里查出的`$result['filename']`再一次入库，结果造成一个二次注入。

利用二次操作进行getshell

那么注入有什么用？
这应该是大家拿到题目，想到的第一个问题。这题明显与getshell有关，源码里包含文件上传、文件改名、文件删除等函数。
我们来一个个分析。

首先upload.php是文件上传的操作，但可见上传处对文件进行了白名单验证：

```
#!php
if(!in_array($path_parts["extension"], ["gif", "jpg", "png", "zip", "txt"])) {
    exit("error extension");
}
复制代码
```

导致我们无法上传恶意文件。

其次是delete.php，这个文件其实是个烟雾弹，删除操作并不能利用。
再次是rename.php，这里明显是getshell的关键。

```
#!php
$result = $db->query("select * from `file` where `filename`='{$req['oldname']}'");
if ($result) {
    $result = $result->fetch();
}

if(!$result) {
    exit("old file doesn't exists!");
} else {
    $req['newname'] = basename($req['newname']);
    $re = $db->exec("update `file` set
                `filename`='{$req['newname']}',
                `oldname`='{$result['filename']}'
                where `fid`={$result['fid']}");
    if(!$re) {
        print_r($db->errorInfo());
        exit;
    }
    $oldname = UPLOAD_DIR . $result["filename"] . $result["extension"];
    $newname = UPLOAD_DIR . $req["newname"] . $result["extension"];
    if(file_exists($oldname)) {
        rename($oldname, $newname);
    }
复制代码
```

最重要的就是后面这5行。
Oldname和newname，有几个特点：

> 1. 后缀相同，都是$result[‘extension’]
> 2. oldname的文件名来自数据库，newname的文件名来自用户输入

首先后缀相同这个特点，就导致getshell似乎难以完成，如果要getshell那么一定要将“非.php”后缀的文件重命名成“.php”的文件。后缀相同怎么重命名？
除非后缀为空！
所以我们的update型注入就开始派上用场了。通过update型注入，我们可以将数据库中extension字段的值改为空，同时也可以控制filename的值，那么等于说我能控制rename函数的两个参数的值，这样getshell就近在咫尺了。

但还有个坑，这里改名的时候检查了文件是否存在：`if(file_exists($oldname))`
我虽然通过注入修改了filename的值，但我upload目录下上传的文件名是没有改的。
因为我利用注入将extension改为空了，那么实际上数据库中的filename总比文件系统中真是的文件名少一个后缀。
那么这里的file_exists就验证不过。怎么办？
简单啊，再次上传一个新文件，这个文件名就等于数据库里的filename的值就好了。

所以最后整个getshell的流程，实际上是一个二次注入 + 二次操作getshell.

具体操作

1.选择文件上传

2.rename造成注入：

3.上传真正包含webshell的文件x.jpg

4.重命名进行getshell：

5.成功

0x01 反序列化+auto_register导致的代码执行

本题考察的是PHP反序列化碰上auto_register导致的安全问题。

找到源码

目标 `http://24caf446e2bb0e659.jie.sangebaimao.com/`
首先扫描发现其包含.git目录，但访问/.git/index发现没有这个文件，可能是被破坏了。
用lijiejie的工具无法还原，但用某些工具还是可以办到的，详见我之前的文章：www.leavesongs.com/PENETRATION…
就不再赘述，用某工具直接还原源码：

getshell

首先通读源码，发现有几个特点：

> 1. 可以上传任意文件，后缀有黑名单检查，文件名是随机字符串md5值
> 2. 数据存储于cookie中，通过php反序列化函数还原并显示

其实考察点比较有意思。
看到common.inc.php里，包含spl_autoload_register函数，这个函数是自动注册类用的，在当今特别是新型的框架（laravel、composer）中常用。
这个函数有个特点，如果不指定处理用的函数，就会自动包含“类名.php”或“类名.inc”的文件，并加载其中的“类名”类。
这就比较有意思了，我们之前的黑名单是不包括“.inc”文件的，所以我们可以按照下面方法进行getshell：

1.上传webshell，后缀为.inc，被重命名为xxxx.inc

2.序列化一个类名为xxxx的类对象

3.将序列化以后的字符串作为cookie，发送到服务器上

4.服务器反序列化这个字符串后，将会自动加载xxxx类，由于之前spl_autoload_register函数注册的方法，会自动加载xxxx.inc，从而造成文件包含漏洞，getshell成功

在网站根目录的flag-1.php中获得第一个flag。

利用本地redis提权

拿到webshell以后，查看一下服务器的一些敏感信息。
比如在phpinfo里看到了，session的处理方式用的redis，并且save_path里暴露了redis的端口和密码：

于是可以利用这段时间比较火的redis写公钥文件进行提权。
直接编写一个redis.php，用php来连接redis，执行redis写公钥的POC：

```
#!php
<?php
$redis = new Redis();
$redis->connect('127.0.0.1', 21821);
$redis->auth("Tat141uIyX8NKU");
$redis->flushall();
$redis->config("SET", "dir", "/root/.ssh/");
$redis->config("SET", "dbfilename", "authorized_keys");
$redis->set("0", "\n\n\nssh-rsa key_pub\n\n\n");
$redis->save();
复制代码
```

连接其ssh端口，直接获取root权限。
读取/root/flag-2.txt获得第二个flag。

0x02 PHP类型与逻辑+fuzz与源代码审计

本题考察了PHP类型与变量的特点，与参赛选手对于一个『不明白』的问题的解决方案（fuzz或阅读源码）。
源码如下

```
#!php
<?php
if(isset($_GET['source'])){
    highlight_file(__FILE__);
    exit;
}
include_once("flag.php");
 /*
    shougong check if the $number is a palindrome number(hui wen shu)
```

```
    shougong check if the $number is a palindrome number(hui wen shu)
 */
function is_palindrome_number($number) {
    $number = strval($number);
    $i = 0;
    $j = strlen($number) - 1;
    while($i < $j) {
        if($number[$i] !== $number[$j]) {
            return false;
        }
        $i++;
        $j--;
    }
    return true;
}
ini_set("display_error", false);
error_reporting(0);
$info = "";
$req = [];
foreach([$_GET, $_POST] as $global_var) {
    foreach($global_var as $key => $value) {
        $value = trim($value);
        is_string($value) && is_numeric($value) && $req[$key] = addslashes($value);
    }
}

$n1 = intval($req["number"]);
$n2 = intval(strrev($req["number"]));
if($n1 && $n2) {
    if ($req["number"] != intval($req["number"])) {
        $info = "number must be integer!";
    } elseif ($req["number"][0] == "+" || $req["number"][0] == "-") {
        $info = "no symbol";
    } elseif ($n1 != $n2) { //first check
        $info = "no, this is not a palindrome number!";
    } else { //second check
        if(is_palindrome_number($req["number"])) {
            $info = "nice! {$n1} is a palindrome number!";
        } else {
            if(strpos($req["number"], ".") === false && $n1 < 2147483646) {
                $info = "find another strange dongxi: " . FLAG2;
            } else {
                $info = "find a strange dongxi: " . FLAG;
            }
        }
    }
} else {
    $info = "no number input~";
}
?>
复制代码
```

在题目上线前，我已经让部分人测试过，当时大家找到了一些解决方法。
之前没有这句话`$req["number"] != intval($req["number"])`，所以大家有很多方法可以解决这个问题，比如1x10、01.1
于是我加了上面这句判断，这样就可以限制这些解法。现在说一下最终得到的三种解决方案。

利用整数溢出绕过

这是最简单的方法，用的是php的整数上限。借用下 @蓝加白 写的writeup（条理清晰，思路很好）。
首先，看一下源代码。发现要找到FLAG，必须要满足以下三个条件：

1. number = intval(number)
2. intval(number) = intval(strrev(number))
3. not a palindorme number

貌似第二个条件和第三个条件冲突了，但是我们可以利用intval函数的限制：
php.net/manual/zh/f…
看一下解释：最大的值取决于操作系统。 32 位系统最大带符号的 integer 范围是 -2147483648 到 2147483647。举例，在这样的系统上， intval('1000000000000') 会返回 2147483647。64 位系统上，最大带符号的 integer 值是 9223372036854775807。
从上面我们可以知道，intval函数还依赖操作系统，很明显测试的环境系统是64位，所以应该选：9223372036854775807。
但有个问题，它的回文数明显小于64位系统的限制，所以我们想到前面加个0；
最终payload: `http://f2ed13418097d206c.jie.sangebaimao.com/?number=09223372036854775807`

利用浮点数精度绕过

这是 @玉林嘎 提出来的解决方案。
我来说一下原理。首先在电脑上测试下面的php代码：

可见，在小数小于某个值（`10^-16`）以后，再比较的时候就分不清大小了，这与php内部储存浮点数的机制有关。
在计算机里，是不能精确表示某个浮点数的。比如1.0，通常情况下储存在计算机里的数值是1.000000000000xxx，是一个十分接近1.0的数。
所以，我们在执行这个if语句的时候`if ($req["number"] != intval($req["number"]))`，会先将右值转换成整数，再与左值比较。而左值是一个浮点数（1.000000000000001），所以右值又会被隐式地强制转换成浮点数1.0
那么1.0和1.000000000000001究竟是否相等呢？
因为我前面说的特性，1.0其实也不是精准的1.0，所以php在比较的时候是不能精准比较浮点数的，所以它会『忽略』比10的-16次方更小的部分，然后就会认为左值和右值相等。

回到CTF中，利用这个特性，我们构造`1000000000000000.0000000000000010`，即可绕过第一个if语句，并且拿到flag。

函数特性导致绕过

这个特性涉及到php『数字类』函数的一个特性。什么函数？包括is_numeric和intval等包含数字判断及转换的函数。
is_numeric为例，我们先来看他的源代码：

可见我画框的部分，is_numeric函数在开始判断前，会先跳过所有空白字符。这是一个特性。
也就是说，`is_numeirc(" \r\n \t 1.2")`是会返回true的。
同理，`intval(" \r\n \t 12")`，也会正常返回12。
这就完成了一半。但有的同学又问了，题目获取$req['number']的时候明明使用trim过滤了空白字符的呀？
我们再看到trim的源码：

掰指头算一下，这里过滤的空白字符和之前跳过的空白字符有什么区别？
少了一个"\f"，嘿嘿。
于是我们可以引入\f（也就是%0c）在数字前面，来绕过最后那个is_palindrome_number函数，而对于前面的数字判断，因为intval和is_numeric都会忽略这个字符，所以不会影响。
最后通过payload: `http://f2ed13418097d206c.jie.sangebaimao.com/?number=%0c121` 拿到第二个flag：