怎样挖掘出属于自己的 php 反序列化链

> 先知社区，先知安全技术社区

Q:
为什么要写这篇文章?

A:
RCTF2020 中的 swoole 一题刺激到我了（，那道题找了两天也找不到链。
再后来第五空间 2020 的那个 laravel 也是找了一天，最后还是靠 phpggc 做了次脚本小子。
我在想为啥我找不到链呢？故有此文。

PS: 虽然写的是一些总结性的东西。但作者也不过找到 5 条框架的链而已，见识还是太少。希望师傅们多多包涵，加以指正。

在使用 php 的反序列化漏洞前需要两个条件

1. 可以进行反序列化的点

2. 合理的 pop chain
   这一对组合拳形成的反序列化漏洞可以进而造成 RCE、文件读写、信息泄露等危害。

本文不会对形成反序列化漏洞的点，进行讲解，其它大师傅已经讲解的十分详细了。
这里就我这两天的挖链经历进行一个总结。

我在挖链的途中总结出以下两点

1. 变量可控
   在危险的函数和结构上的可控变量要尽可能的多

2. 扩大影响
   尽可能的去寻找可以扩大攻击面的结构与方法

我们接下来的 pop chain 构造便一直基于这两点

序列化是将对象的属性进行格式的转换，但不会包括方法。所以如果想要反序列化达成恶意的操作必须需要方法的执行。
对于起点来说，我们自然是要找到可以自动调用的方法。常见的可以自动调用的方法便是魔术方法。魔术方法的介绍有很多，这里就不详细介绍了。

目前只有两个魔术方法可以被使用

1. `__destruct`

2. `__wakeup`

其中，最为常用的魔术方法是 `__destruct` ，其特性是对象被销毁前被调用。从系统结构的角度讲，其最常见的场景是关闭某些功能。比如关闭文件流，更新数据等。但从反序列化的角度讲，其特殊的使用场景，代表在这个方法内可能会调用类内的其它方法。

而另一个 `__wakeup` 方法就不太常用了，其特性是反序列化时进行调用。那么可以想象开发人员在对其进行编写时，可能会将其作为一个 “进行反序列化时属性合法性校验的” 方法。
最经典的就是 `GuzzleHttp` 包中的 `GuzzleHttp\Psr7\FnStream` 类，其内部存在大量变量可控的危险

函数。但以下这一个方法就直接避免了这个方法被恶意使用。

```
public function __wakeup()
{
    throw new \LogicException('FnStream should never be unserialized');
}
```

当然也不是没有使用了 `__wakeup` 的链，只不过从各个方面来讲， `__destruct` 确实更好用一些。

接下来再看一个例子。
phpggc (https://github.com/ambionics/phpggc) 是 github 上的一个项目，其存储着大量反序列化链，可以说是反序列化的武器库。
其存储了大量的 laravel 框架的 RCE 反序列化链，仔细观察发现一共 6 条反序列化链，5 条都使用了同一个类作为起点，还有一条也间接调用了此类。其便是 `PendingBroadcast.php` 下的 `__destruct` 方法

```
public function __destruct()
{
    $this->events->dispatch($this->event);
}
```

为什么这个方法会变成公交车呢?
我认为共 3 点

1. 可自动调用
2. 参数可控
3. 攻击面广
   自动调用自然不必多说， `__destruct` 方法嘛。但值得注意的是其参数，和结构。

我们可以看到 `$this->events` 和 `$this->event` 都是可控的，这意味着我们的链可以有两条走向。
其一是将 `$this->events` 赋值为没有 `dispatch` 方法的实例，来调用其 `__call` 方法。
其二是去调用各种类中 `dispatch` 方法，如果有的 `dispatch` 中有危险的函数或者结构，那就考虑使用它。
这样就大幅扩大了我们的攻击面。

所谓的跳板，就是在方法和方法、结构和结构、方法和结构之间的跳跃。

常见的例子是一些字符串函数，例如 `trim` ，如果其参数可控，我们将其赋值为存在 `__toString` 方法的对象即可调用这个方法。

还有类似于 `call_user_func($this->test);` 或者 `$test();` 这种只能调用没有参数的函数的结构。出来简单的调用 phpinfo 以外，我们也可以考虑将变量赋值为 `[(new test), "aaa"]` 这样的一个数组。就可以调用 `test` 类中的 `aaa` 公共方法。

再者，就是 `new $test1($test2, $test3);` 这样的结构也可以调用 `__construct` 方法。或者像 RCTF2020-swoole 一题一样，新建一个 PDO 对象来进行 mysql 的 load file。

总之，就是不计一切代价扩大链的可能性，为寻找到可以利用的方法提供机会。

终点在我看来有两类

1. 危险动态调用

2. 危险函数

动态调用就是像 `($this->a)($this->b)` 或者 `$this->a[0]($this->b)` 或者这样的危险动态调用。

危险函数，就是根据目的寻找需要的函数。如要 RCE，则寻找类似

于 `call_user_func` ， `array_walk` 这样的会进行函数调用的函数。如要 FW，则寻找 `file_put_content` 这样的函数...

这里稍微讲一讲 Yii2 框架的链吧。当时搜了一波文章好像也没有。

环境准备我就不详细写了

```
composer create-project yiisoft/yii2-app-basic app
```

composer+docker+vscode 一把梭

Yii2/RCE1

首先全局搜索 `__destruct` 和 `__wakeup` 这两个魔术方法。可以使用 grep 命令 `grep -A 10 -rn "__destruct"` 。或者直接使用 vscode 的全局搜索也可。

最后我将其定位在 `yii\db\BatchQueryResult` 类中的 `__destruct`

```
public function __destruct()
    {
        $this->reset();
    }

    public function reset()
    {
        if ($this->_dataReader !== null) {
            $this->_dataReader->close();
        }
        $this->_dataReader = null;
        $this->_batch = null;
        $this->_value = null;
```

```
        $this->_key = null;
    }
```

可以看到，就像我刚才说的一样，这里既可以调用 `__call` 方法，也可以调用 `close` 方法。

在搜了一波 `__call` 方法感觉没戏后，这里我选择调用了 `close` 方法
全局搜 `grep -A 10 -rn "function[[:space:]]close"` ，或者 vscode

这里我选择了 `yii\web\DbSession` 类中的 `close` 方法

```
public function close()
    {
        if ($this->getIsActive()) {
            // prepare writeCallback fields before session closes
            $this->fields = $this->composeFields();
            YII_DEBUG ? session_write_close() : @session_write_close();
        }
    }
```

首先调用了父类的 `getIsActive` 方法

```
public function getIsActive()
    {
        return session_status() === PHP_SESSION_ACTIVE;
    }
```

可以看到其对会话的状态进行了一个判别。这里我意外发现，只有当 Yii 的 debug 和 gii 这两个默认扩展都存在（不一定要开启）时，这里返回 true。否则返回 false。这里我还不知道为什么，希望有师傅可以解答...

这里算是这条链唯一的缺憾了吧...
总之返回 true 后继续调用了接口类 `yii\web\MultiFieldSession` 的 `composeFields` 方法

```
protected function composeFields($id = null, $data = null)
    {
        $fields = $this->writeCallback ? call_user_func($this->writeCallback, $this) : [];
        if ($id !== null) {
            $fields['id'] = $id;
        }
        if ($data !== null) {
            $fields['data'] = $data;
        }
        return $fields;
    }
```

芜湖，发现了个可控的 `call_user_func` ，但可惜参数无法控制，传入一个对象为参数的可用函数也不太多。那就像我刚才所说，赋值为 `[(new test), "aaa"]` 这样的一个数组。就可以调用 `test` 类中的 `aaa` 公共方法。

那么有没有这样的公共方法?

`grep -A 10 -rn "public[[:space:]]function[[:space:]].*\(\)"`

最后我找到了 `yii\rest\IndexAction` 中的 `run` 方法

```
public function run()
    {
        if ($this->checkAccess) {
```

```
            call_user_func($this->checkAccess, $this->id);
        }

        return $this->prepareDataProvider();
    }
```

这下两个都可控了。

(https://xzfile.aliyuncs.com/media/upload/picture/20200731022748-5e8d5d10-d292-1.png)

Yii2/RCE2

因为上一条链受扩展影响，所以打算再找一条
这次我定位到 `\Symfony\Component\String\UnicodeString` 的 `__wakeup` 类

```
public function __wakeup()
    {
        normalizer_is_normalized($this->string) ?: $this->string = normalizer_normalize($this->strin
g);
    }
```

我看了一下 这个函数，其要求参数是字符串，参数又可控，那就调

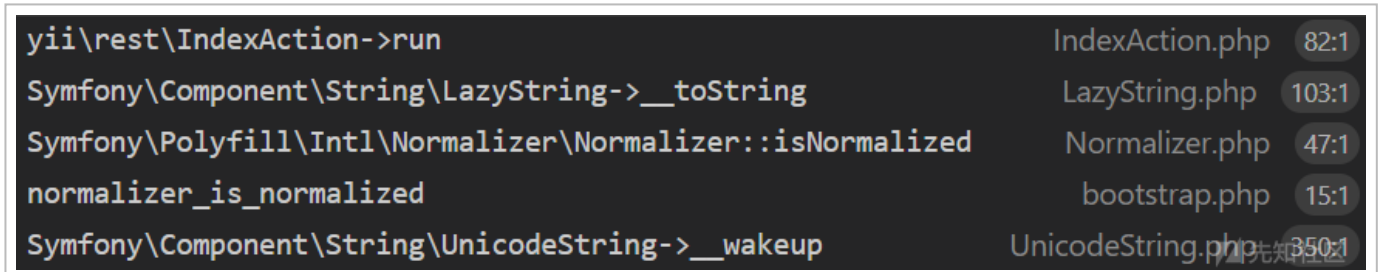
找着了一个 `normalizer_is_normalized` 这个函数，其要求参数是字符串。参数又可控，那就调用 `__toString` 方法吧。

最后我找到了 `\Symfony\Component\String\LazyString` 的 `__toString` 方法

```
public function __toString()
    {
        if (\is_string($this->value)) {
            return $this->value;
        }

        try {
            return $this->value = ($this->value)();
        } catch (\Throwable $e) {
        //...
        }
    }
```

啊这... 结合最后一部分一打不就完事了?
喜加一



(https://xzfile.aliyuncs.com/media/upload/picture/20200731022716-4babc2f4-d292-1.png)

动态调用与危险函数

1. 在写到动态调用和危险函数时，务必对变量和方法进行回溯。查看变量是否是可控的。

2. 在容许的情况下，使用静态属性进行动态调用可以防止可控变量调用危险函数。

3. 在调用 `$this->aaa->bbb()` 这样类似的结构前可以利用 `instanceof` 进行检查，查看其是否是期望调用的类。

方法

1. 注意尽量少的在魔法方法中写入可以调用大量其它方法的方法。尤其是 `__destruct` 和 `__wakeup` 。

2. 注意在公共且不需要参数的方法中不要直接调用危险函数和动态调用。

3. 在不需要 `__wakeup` 方法且类没必要序列化时，可以考虑使用 `__wakeup` 阻止反序列化。

最最最最最重要的

不要让 unserialize 和文件类函数用户可控！！！
不要让 unserialize 和文件类函数用户可控！！！
不要让 unserialize 和文件类函数用户可控！！！

emmm ~~感觉写了一篇水文~~

那就这样，这里放上我 fork 的 phpggc (https://github.com/AFKL-CUIT/phpggc)，有想学习上面几条链的师傅可以看看

如有错误还请指出！