源代码采用了explode函数，将file字符串根据"."进行了分组

魔改数据包，根据源码将save_name分组

```php
$is_upload = false;
$msg = null;
if(!empty($_FILES['upload_file'])){
    //检查MIME
    $allow_type = array('image/jpeg','image/png','image/gif');
    if(!in_array($_FILES['upload_file']['type'],$allow_type)){
        $msg = "禁止上传该类型文件!";
    }else{
        //检查文件名
        $file = empty($_POST['save_name']) ? $_FILES['upload_file']['name'] : $_POST['save_name'];
        if (!is_array($file)) {
            $file = explode('.', strtolower($file));
        }

        $ext = end($file);
        $allow_suffix = array('jpg','png','gif');
        if (!in_array($ext, $allow_suffix)) {
            $msg = "禁止上传该后缀文件!";
        }else{
            $file_name = reset($file) . '.' . $file[count($file) - 1];
            $temp_file = $_FILES['upload_file']['tmp_name'];
            $img_path = UPLOAD_PATH . '/' .$file_name;
            if (move_uploaded_file($temp_file, $img_path)) {
                $msg = "文件上传成功! ";
                $is_upload = true;
            } else {
                $msg = "文件上传失败! ";
            }
        }
    }
}else{
    $msg = "请选择要上传的文件! ";
}
```

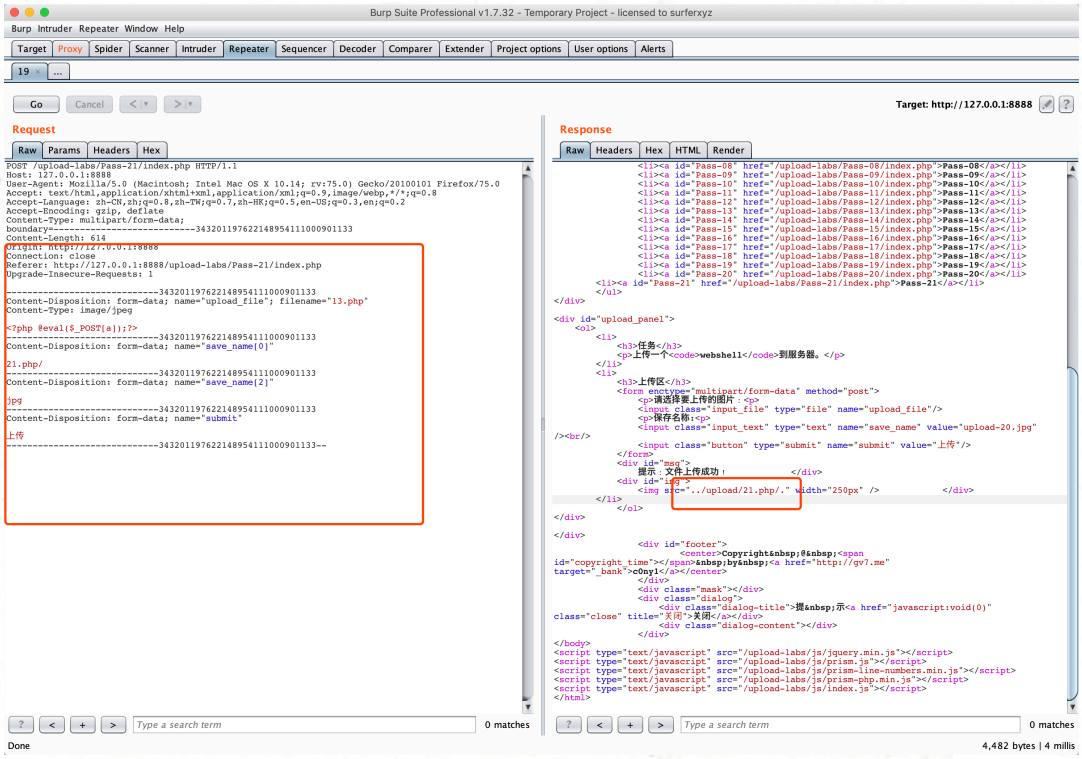
由于这一行的存在，将save_name分为三部分，根据源代码红框部分，会对第三部分进行白名单检查，然后将第一部分和第三部分组合起来（但是我们将第二部分设为空之后，判断个数的count函数实际上就只判断为两个，所以file[1]就恰好是第二个空的部分，若不将第二个设为空，那么count函数返回值为3，就映射到了第三部分，我在末尾实验也证明了这一点）

第一部分为21.php/，为什么要加这个/呢，因为根据代码后面会跟一个"."，由move_upload_file函数漏洞，/.就会被忽略掉从而变为21.php

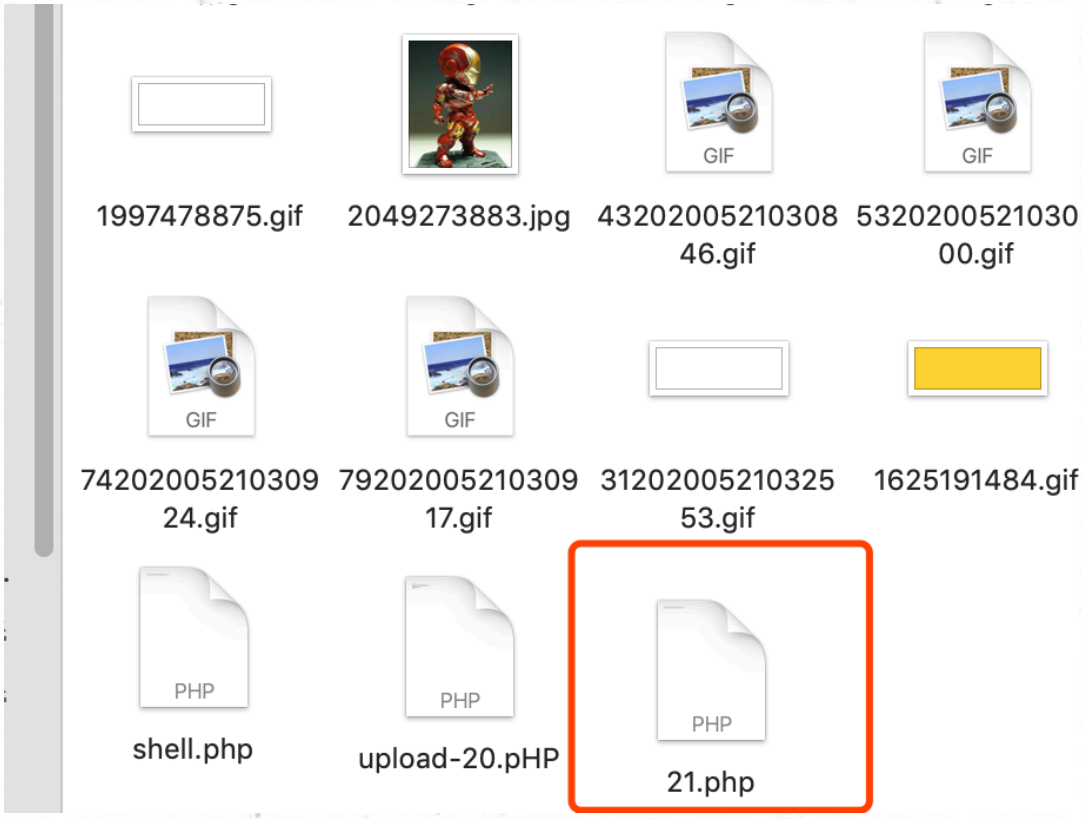
第二部分为空，但是由于我这个环境不能用%00截断，因此需要手动分组将其设为空，方法就是不传递它，见下图，只传递0号元素和2号元素，这样使count变为2，从而恰好file[2-1]为file[1]就是空

第三部分为jpg，是为了配合白名单检查

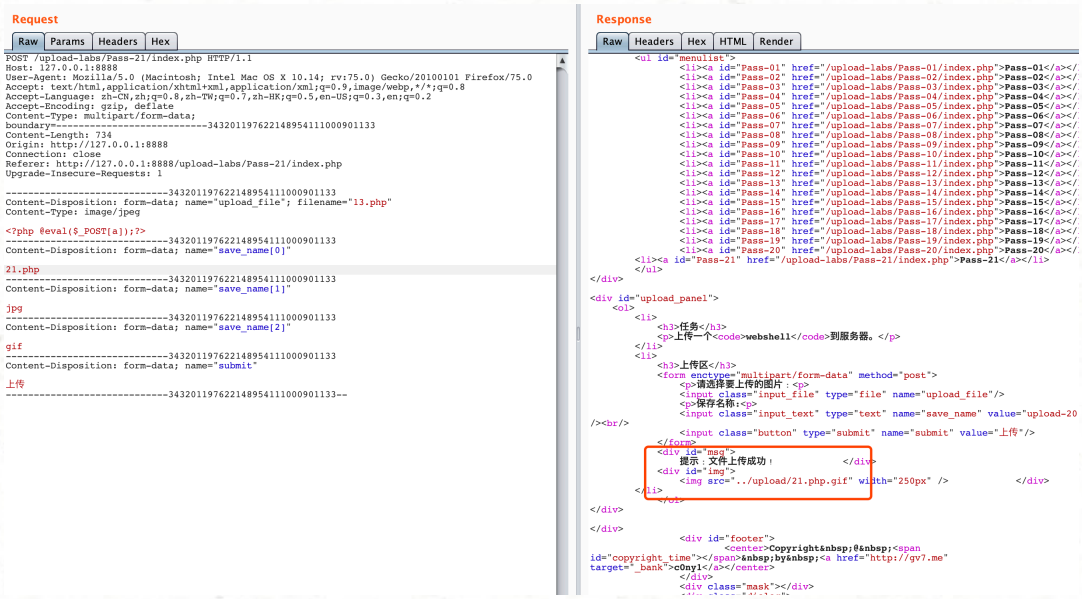
另外由于源码的检查，需要将MIME改为image/jpg



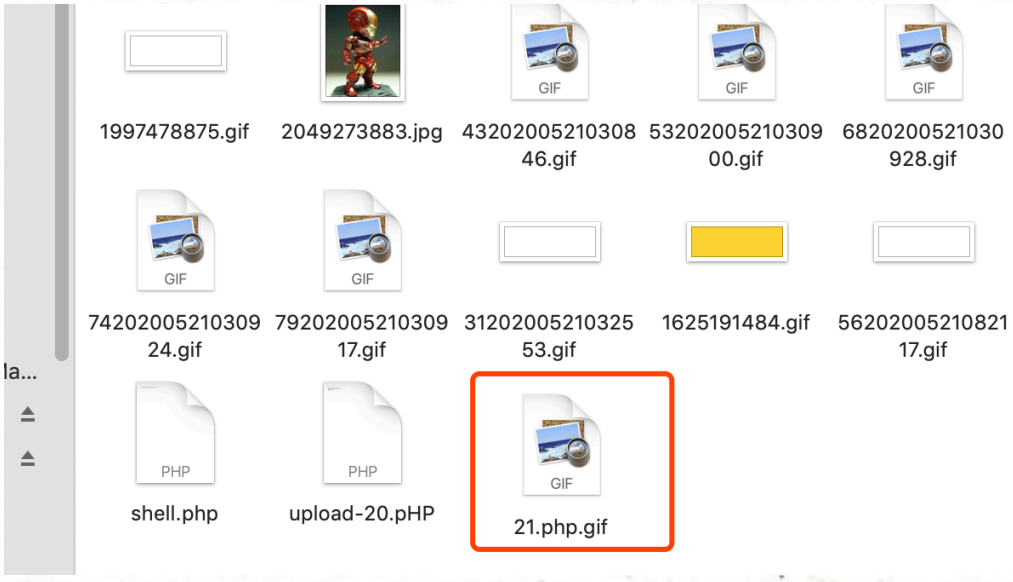
左侧只传递0、2两个元素，右侧可以看到存储路径为..upload/21.php/.，然后/.被自动忽略



验证一下若第二部分不为空会怎么样



果然组合的是第一部分+第三部分



上传的文件也证明了这一点

代码审计很重要呀！