Upload Attack Framework

Version 1.0

CasperKid [Syclover][Insight-Labs]

2011/10/6

[*] Team : Syclover Security Team & Insight Labs
[*] Web : www.myclover.org & www.insight-labs.org
[*] Blog : hi.baidu.com/hackercasper
[*] Mail : casperkid.syclover@gmail.com

前言

这篇 paper 还是断断续续憋了我两个月左右，想把整个攻击能抽象出一个体系出来，所以一直拖了很久才最终成型，个中艰辛还是在自己能体会。原本 paper 是投到 webzine0x06，后来刺总(axis)说 webzine0x06 要 11 月才发布，很多朋友又在催，就提前公开这篇 paper 了。

在现在越来越安全的体系下，sql injection 这类漏洞已经很难在安全性较高的站点出现，比如一些不错的.NET 或 JAVA 的框架基本上都是参数化传递用户输入以及其他一些能防御 SQL 注入的 API，直接封死注入攻击。在非 php 的 web 安全中出现概率很大而且威力也很大的攻击主要有两种，第一种是 sql injection，第二种便是上传攻击。（php 的还有本地/远程文件包含或代码注入漏洞等)。

通常 web 站点会有用户注册功能，而当用户登入之后大多数情况下都会存在类似头像上传、附件上传一类的功能，这些功能点往往存在上传验证方式不严格的安全缺陷，是在 web 渗透中非常关键的突破口，只要经过仔细测试分析来绕过上传验证机制，往往会造成被攻击者直接上传 web 后门，进而控制整个 web 业务的控制权，复杂一点的情况是结合 web server 的解析漏洞来上传后门获取权限。

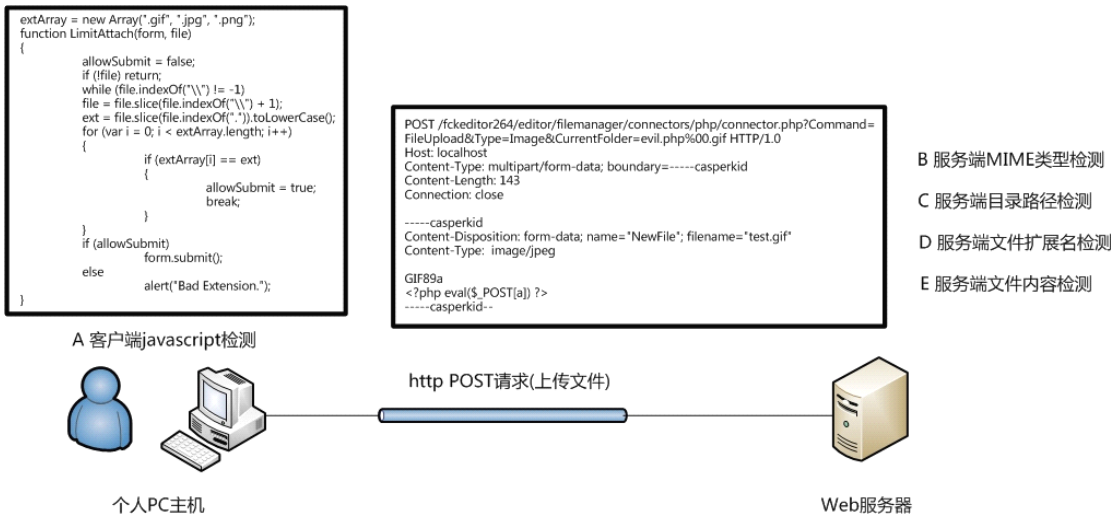
这篇 paper 或许还并不算完善，但在分类总结上还是比较全面，我看过 OWASP 和 Acunetix 对上传攻击的分类，很多方面并不够全面。在写 paper 这段时间，因为也在学习其他东西，比较忙，现在才把这篇 paper 算写了一个相对来说比较完整的版本，麻雀虽小，五脏俱全，之中有不足之处，希望大家有类似经验的提出来，以便我能更加完善这篇 paper，也让大家的交流产生更大的价值。

还有关于这个 framework，当时看有部分朋友有些误解，以为有这个 framework，就一定能挖到 exp 之类，为了不误导大家，在这里还是做个说明。我的本意，只是为了提供一种相对来说系统的分析方法，比如给你一份 upload 的源码或者一个 website，你怎么来系统地进行白盒/黑盒分析它，并从流程中找出它的逻辑漏洞或其他漏洞来，而不是去猜它会有什么漏洞。更高层次的渗透不应该是去猜测目标，而是通过好的分析和总结能确定甚至预测目标，要自己能心里真正清楚目标的信息，是什么版本，有什么漏洞，一次就成功，而不是我猜下这样是否能成功的侥幸心理。渗透不是猜测，而是结合社会工程学收集到足够多的信息，明确甚至预测目标的环境，然后不做过多的动作，而让自己最大化隐藏完成渗透测试。

在写 paper 时，保存的图像分辨率都比较高，大家用 100%分辨率观看时可能不太清晰，需要看图像的细节时，推荐大家用 200%-400%分辨率来观看即可。

目录概览

0x00 上传检测流程概述



通常一个文件以 HTTP 协议进行上传时，将以 POST 请求发送至 web 服务器
web 服务器接收到请求后并同意后，用户与 web 服务器将建立连接，并传输 data

而一般一个文件上传过程中的检测如下图红色标记部分：

A 客户端 javascript 检测 (通常为检测文件扩展名)
B 服务端 MIME 类型检测 (检测 Content-Type 内容)
C 服务端目录路径检测 (检测跟 path 参数相关的内容)
D 服务端文件扩展名检测 (检测跟文件 extension 相关的内容)
E 服务端文件内容检测 (检测内容是否合法或含有恶意代码)

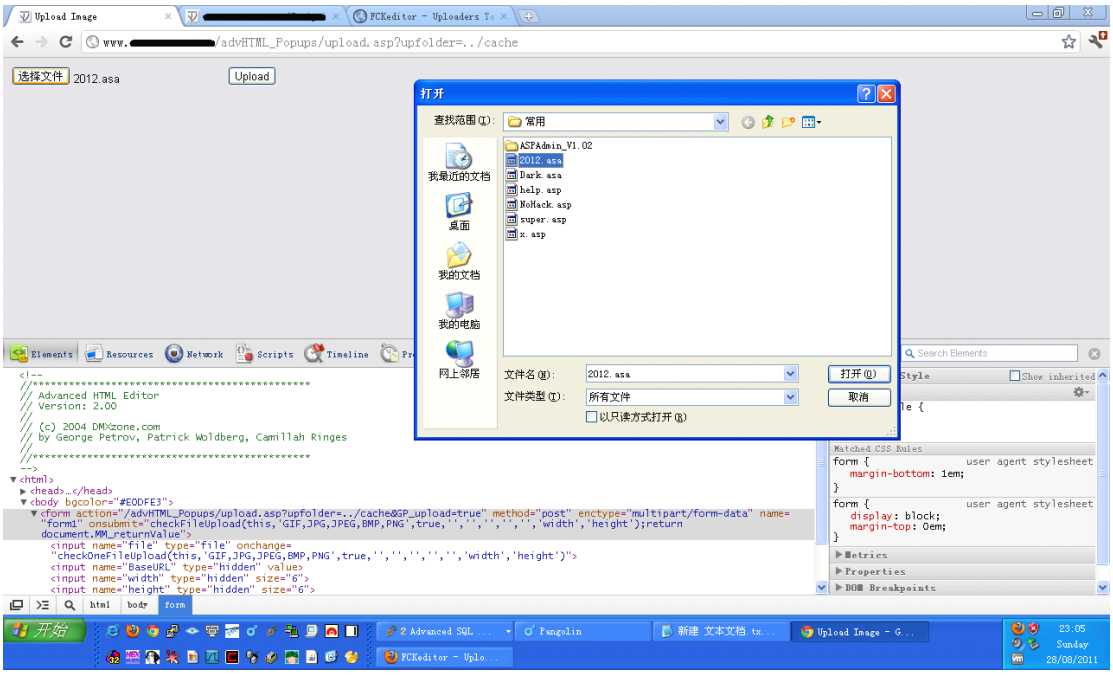
随后本文将对这些检测如何绕过攻击进行详细的讲解

0x01 客户端检测绕过(javascript 检测)

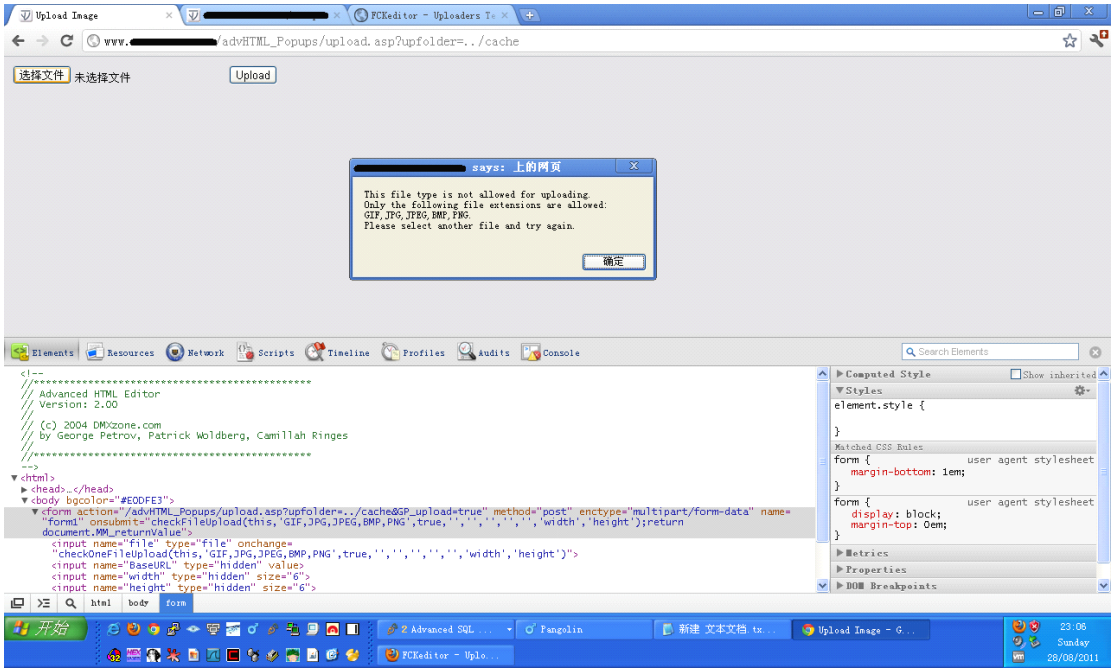
这类检测通常在上传页面里含有专门检测文件上传的 javascript 代码
最常见的就是检测扩展名是否合法

下面将给出具体实例

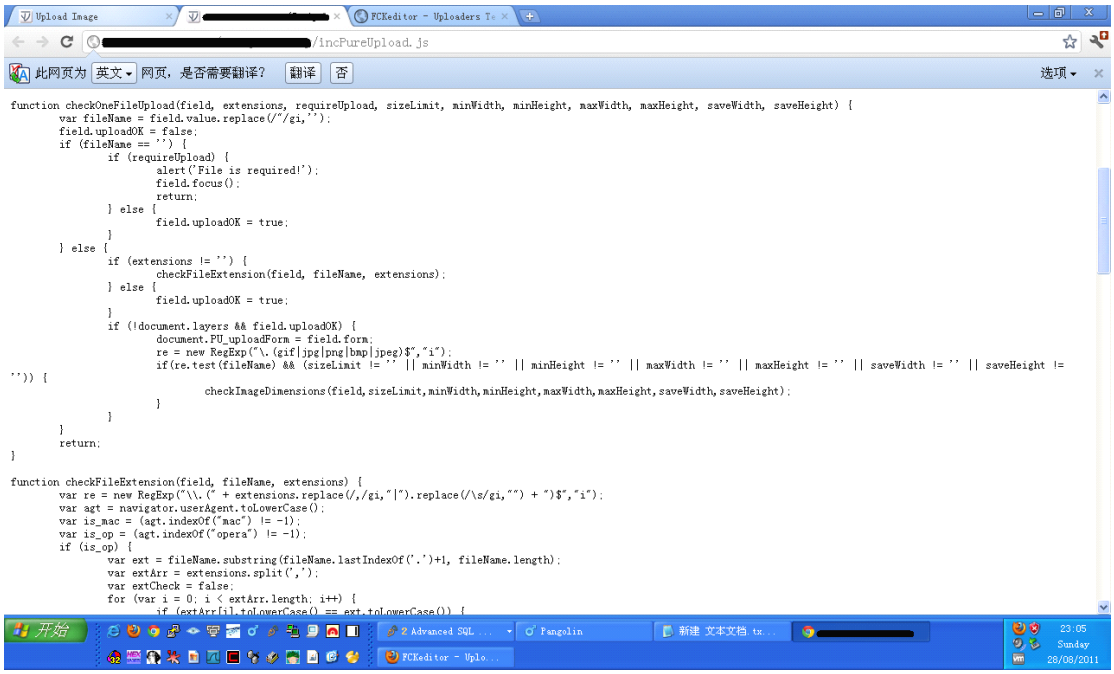
打开 http 反向代理工具 burp
先随便点击浏览选择文件 2012.asa



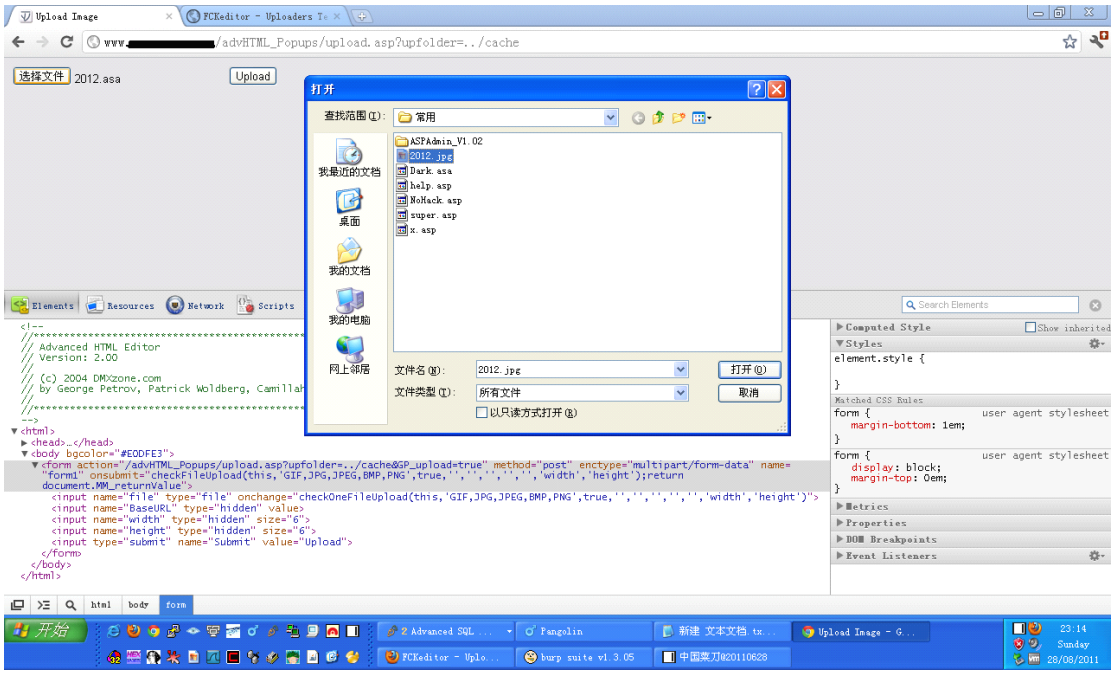
但是刚点击“打开”，而这时也并没点击 Upload
burp 里也还没出现任何内容，便弹出了一个警告框



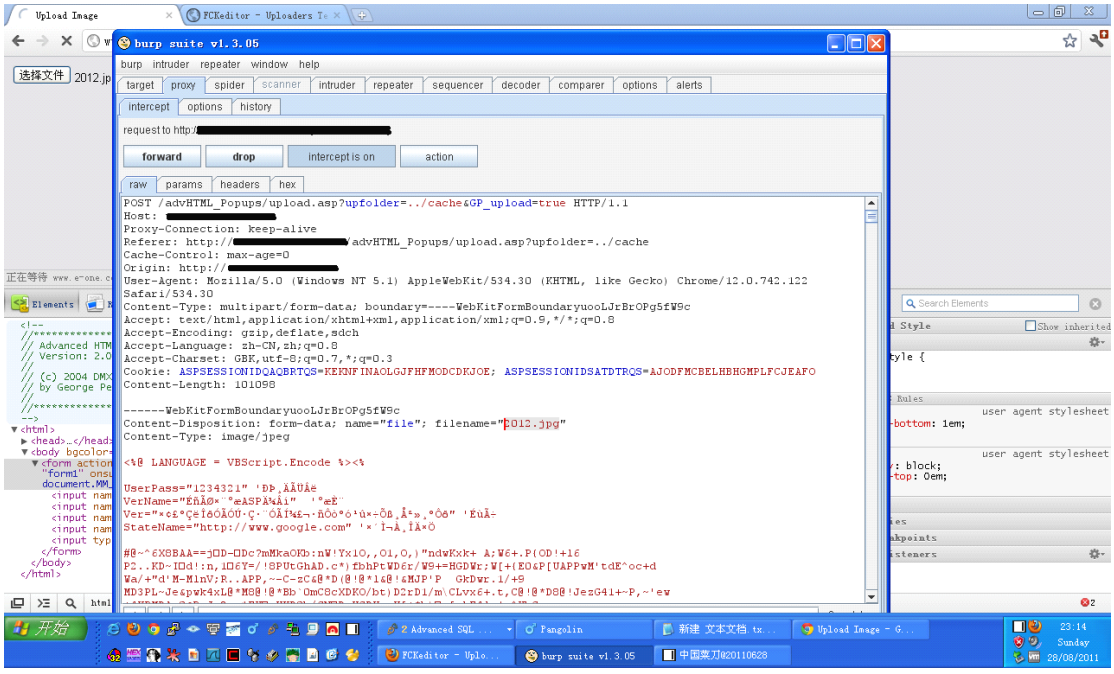
这种情况一看就知道是个客户端 javascript 检测 (因为没有流量经过 burp 代理)
下图是该上传页面负责检测扩展名合法性的 javascript 代码



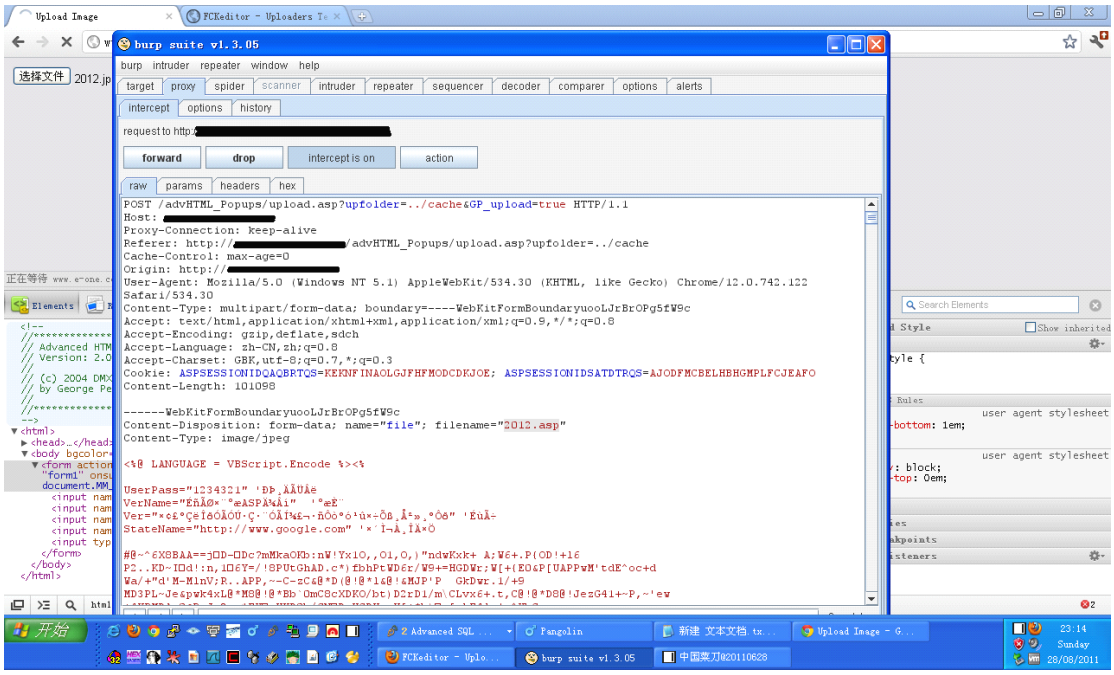
我们可以用 firebug 之类的插件把它禁掉或者通过 burp 之类的代理工具进行绕过提交
这里我将用 burp 进行代理修改
先将文件扩展名改成 jpg



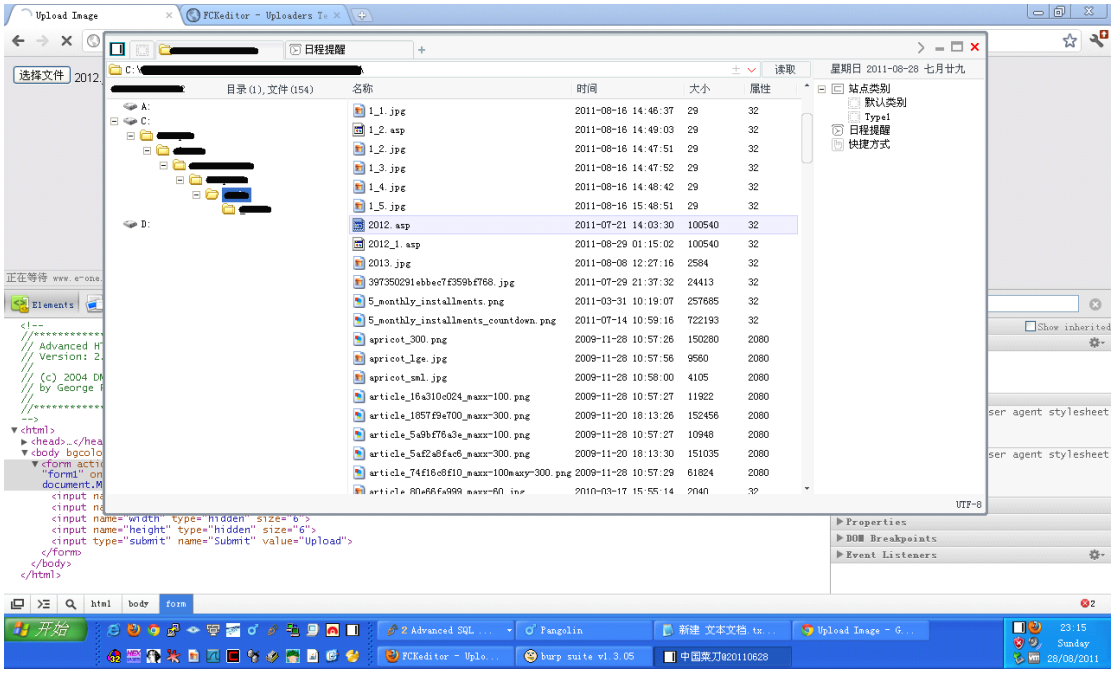
然后点击 Upload
现在在 POST Packet 里文件名字段的值是 2012.jpg



在 burp 里将 filename="2012.jpg" 改成 filename="2012.asp"



然后继续 forward 上传
最后可以看到 asp 成功上传



0x02 服务端检测绕过(MIME 类型检测)

因为没有找到合适的案例源码程序，就找了个模拟代码讲解下
更具体的找到了一个渗透实例，也是利用的 MIME 类型上传绕过漏洞
地址是: http://www.heibai.net/articles/hacker/ruqinshili/2011/0325/13735.html

下面是找到的一个模拟检测代码

假如服务器端上的 upload.php 代码如下

```
<?php
    if($_FILES['userfile']['type'] != "image/gif") { //检测Content-type
    echo "Sorry, we only allow uploading GIF images";
    exit;
    }
    $uploaddir = 'uploads/';
    $uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
    if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
    echo "File is valid, and was successfully uploaded.\n";
    } else {
    echo "File uploading failed.\n";
    }
?>
```

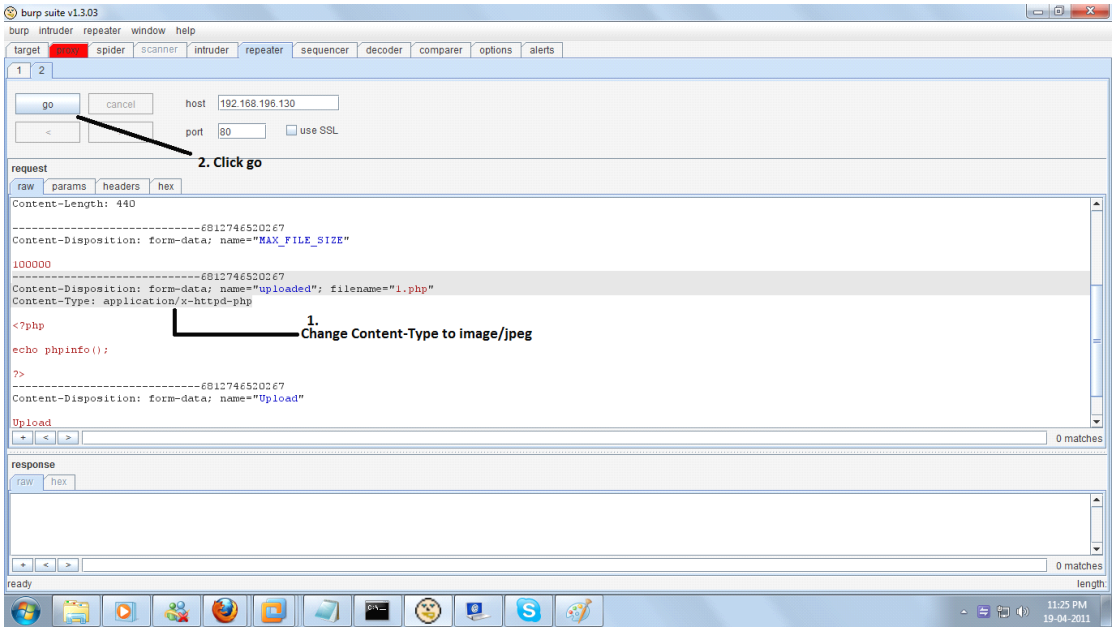
然后我们可以将 request 包的 Content-Type 修改

POST /upload.php HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: localhost
User-Agent: libwww-perl/5.803
Content-Type: multipart/form-data; boundary=xYzZY
Content-Length: 155
--xYzZY
Content-Disposition: form-data; name="userfile"; filename="shell.php"
Content-Type: image/gif (原为 Content-Type: text/plain)
<?php system($_GET['command']);?>
--xYzZY--

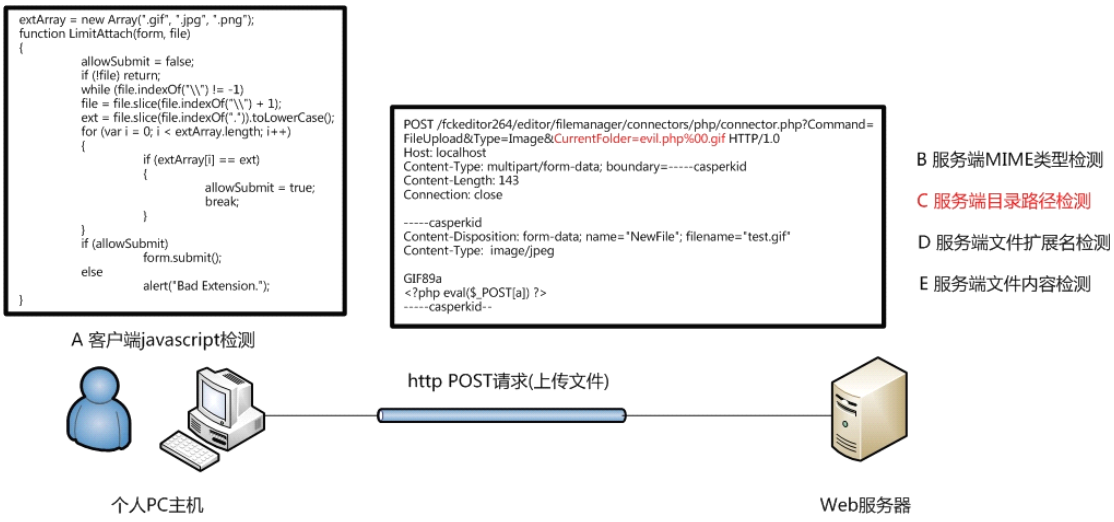
得到服务端的应答

HTTP/1.1 200 OK
Date: Thu, 31 May 2011 14:02:11 GMT
Server: Apache
Content-Length: 59
Connection: close
Content-Type: text/html
<pre>File is valid, and was successfully uploaded.</pre>

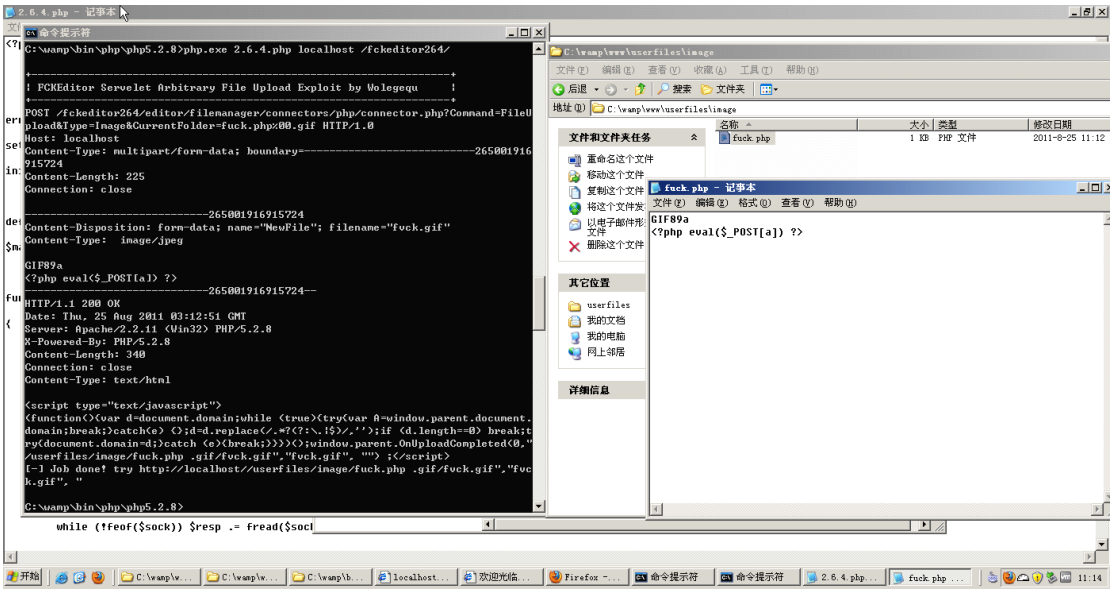
可以看到我们成功绕过了服务端 MIME 类型检测
像这种服务端检测 http 包的 Content-Type 都可以用这种类似的方法来绕过检测



0x03 服务器检测绕过(目录路径检测)

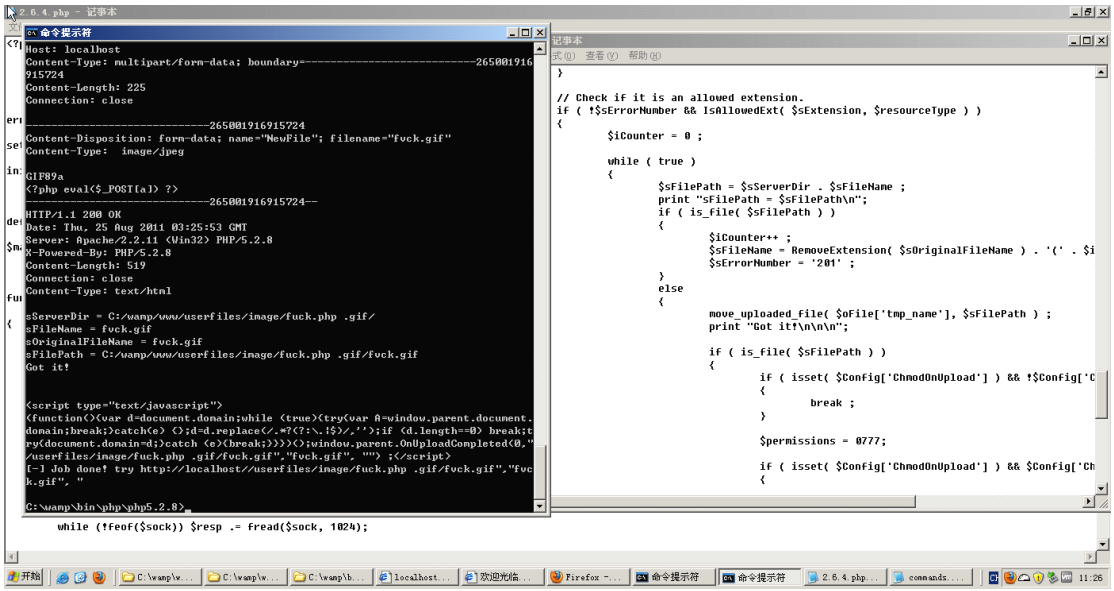


目录路径检测，一般就检测路径是否合法，但稍微特殊一点的都没有防御。
比如比较新的 fckeditor php <= 2.6.4 任意文件上传漏洞
地址: http://www.wooyun.org/bugs/wooyun-2010-01684
效果如下



漏洞成因是因为对目录路径的检测不够严谨而导致可以用 0x00 截断进行攻击

可以看到图中，在最后将要进行写文件之前的变量状态
sFilePath = C:/wamp/www/userfiles/image/fuck.php .gif/fvck.gif (.php 后面是 0x00)
当右图执行 move_uploaded_file($oFile['tmp_name'], $sFilePath) 这个函数时
1. 先将 sFilePath 写入到指定位置，但是底层操作应该是调用的类似于 c 语言，遇到 0x00 会自动截断，所以真正写入的实际地址是 C:/wamp/www/userfiles/image/fuck.php
2. 把原本 fvck.gif 里的内容(目前应该存在临时文件，类似于 C:/Temp/phpf3at7b 这样的文件)然后把 C:/Temp/phpf3at7b 的内容写入到 C:/wamp/www/userfiles/image/fuck.php 里
这样便获得了我们所想要的 webshell



问题出在了 io.php 里的 ServerMapFolder 函数

```
69  function ServerMapFolder( $resourceType, $folderPath, $sCommand )
70  {
71      // Get the resource type directory.
72      $sResourceTypePath = GetResourceTypeDirectory( $resourceType, $sCommand ) ;
73
74      // Ensure that the directory exists.
75      $sErrorMsg = CreateServerFolder( $sResourceTypePath ) ;
76      if ( $sErrorMsg != '' )
77          SendError( 1, "Error creating folder \"{$sResourceTypePath}\" ({$sErrorMsg})" ) ;
78
79      // Return the resource type directory combined with the required path.
80      return CombinePaths( $sResourceTypePath , $folderPath ) ;
81  }
```

当 POST 下面的 URL 的时候
/fckeditor264/filemanager/connectors/php/connector.php?Command=FileUpload&Type=Image&CurrentFolder=fuck.php%00.gif HTTP/1.0
CurrentFolder 这个变量的值会传到 ServerMapFolder($resourceType, $folderPath, $sCommand) 中的形参 $folder 里，而 $folder 在这个函数中并没做任何检测，就被 CombinePaths()了

0x04 服务端检测绕过(文件扩展名检测)

对于扩展名检测不强的，时常还可以结合目录路径攻击
比如 filename="test.asp/evil.jpg" 之类

- 黑名单检测

黑名单的安全性比白名单的安全性低很多，攻击手法自然也比白名单多
一般有个专门的 blacklist 文件，里面会包含常见的危险脚本文件
例如 fckeditor 2.4.3 或之前版本的黑名单

```
44    ConfigDeniedExtensions.Add  "File",
      "html|htm|php|php2|php3|php4|php5|phtml|pwml|inc|asp|aspx|ascx|jsp|cfm|
      cfc|pl|bat|exe|com|dll|vbs|js|reg|cgi|htaccess|asis|sh|shtml|shtm|phtm"
```

1. 文件名大小写绕过
   用像 AsP，pHp 之类的文件名绕过黑名单检测
2. 名单列表绕过
   用黑名单里没有的名单进行攻击，比如黑名单里没有 asa 或 cer 之类
3. 特殊文件名绕过
   比如发送的 http 包里把文件名改成 test.asp. 或 test.asp_(下划线为空格)，这种命名方式在 windows 系统里是不被允许的，所以需要在 burp 之类里进行修改，然后绕过验证后，会被 windows 系统自动去掉后面的点和空格，但要注意 Unix/Linux 系统没有这个特性。
4. 0x00 截断绕过
   在扩展名检测这一块目前我只遇到过 asp 的程序有这种漏洞，给个简单的伪代码

```
name = getname(http request) //假如这时候获取到的文件名是 test.asp .jpg(asp 后面为 0x00)
type = gettype(name) //而在 gettype()函数里处理方式是从后往前扫描扩展名，所以判断为 jpg
if (type == jpg)
    SaveFileToPath(UploadPath.name, name) //但在这里却是以 0x00 作为文件名截断
                                          //最后以 test.asp 存入路径里
```

5. .htaccess 文件攻击
   配合名单列表绕过，上传一个自定义的.htaccess，就可以轻松绕过各种检测
6. 解析调用/漏洞绕过
   这类漏洞直接配合上传一个代码注入过的非黑名单文件即可，再利用解析调用/漏洞

- 白名单检测

白名单相对来说比黑名单安全一些，但也不见得就绝对安全了

1. 0x00 截断绕过
   用像 test.asp%00.jpg 的方式进行截断，属于白名单文件，再利用服务端代码的检测逻辑漏洞进行攻击，目前我只遇到过 asp 的程序有这种漏洞
2. 解析调用/漏洞绕过
   这类漏洞直接配合上传一个代码注入过的白名单文件即可，再利用解析调用/漏洞

- .htaccess 文件攻击

无论是黑名单还是白名单
再直接点就是直接攻击.htaccess 文件
(其实目前我只见过结合黑名单攻击的，在后面的攻击分类里，我会把它归到黑名单绕过攻击里。但网上是把这个单独分类出来的，可能别人有一些我不知道的方式和技巧吧，所以在这里我也暂时保留这个单独分类)

在 PHP manual 中提到了下面一段话
move_uploaded_file section, there is a warning which states
‘If the destination file already exists, it will be overwritten.’
如果 PHP 安全没配置好
就可以通过 move_uploaded_file 函数把自己写的.htaccess 文件覆盖掉服务器上的
这样就能任意定义解析名单了

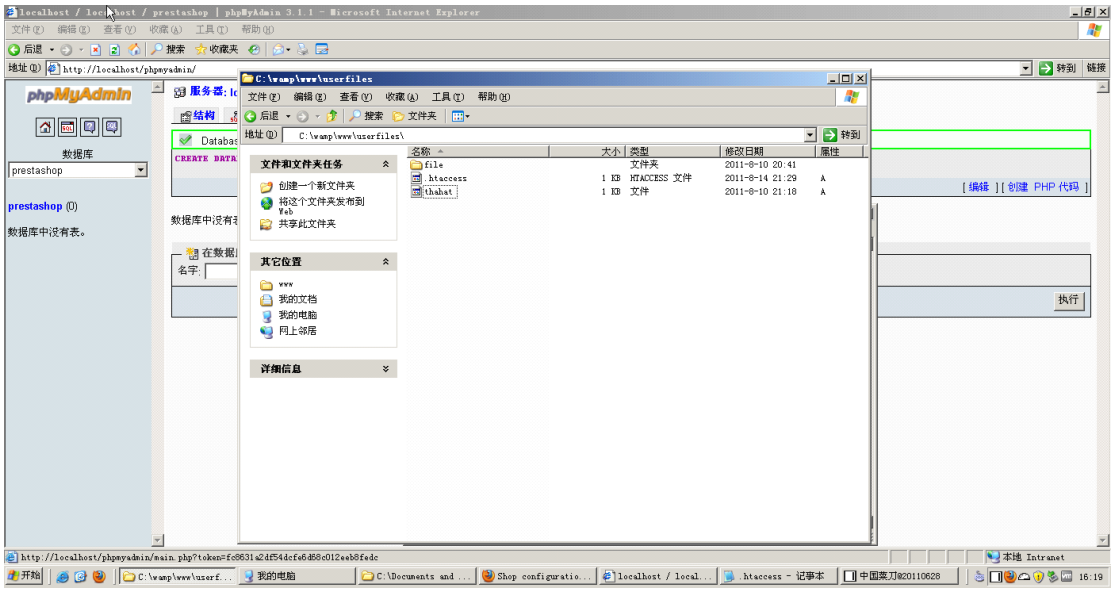
来做个小实验，先描述下效果：

通过一个.htaccess 文件调用 php 的解析器去解析一个文件名中只要包含"haha"这个字符串的任意文件，所以无论文件名是什么样子，只要包含"haha"这个字符串，都可以被以 php 的方式来解析，是不是相当邪恶，一个自定义的.htaccess 文件就可以以各种各样的方式去绕过很多上传验证机制

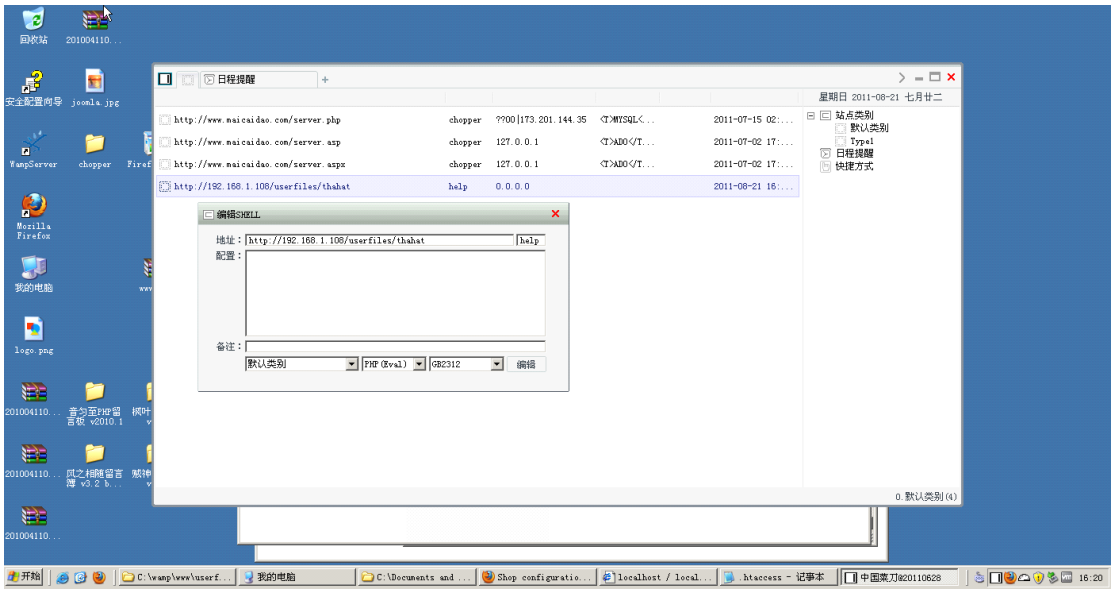
建一个.htaccess 文件，里面的内容如下

```
<FilesMatch "haha">
SetHandler application/x-httpd-php
</FilesMatch>
```

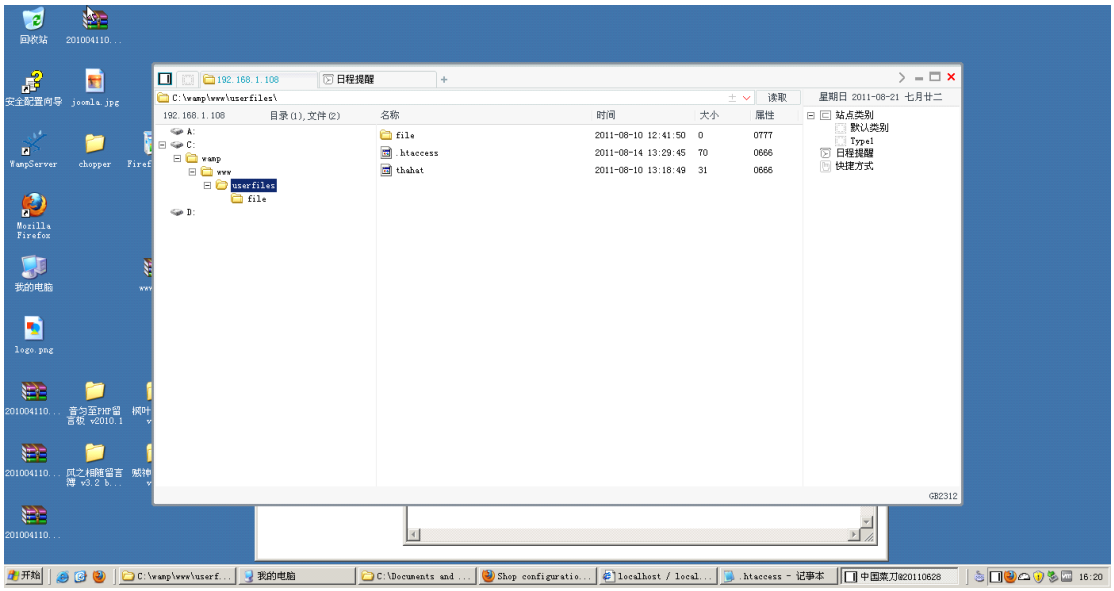
同目录有个我们上传一个只有文件名并包含字符串"haha"，但是却无任何扩展名的文件
里面的内容是 php 一句话木马



然后我们用中国菜刀去连接测试
看看结果是否如我们预期一样 :)
在中国菜刀里进行配置



然后连接过去
从图片上可以看出，结果如我们预期的一样 :)



所以一个可以由 hacker 掌控的.htaccess 文件是非常邪恶的~
基本上可以秒杀各种市面上的上传验证检测 :) (内容检测除外)
从实际环境来说，我个人接触过的，一般是配合黑名单攻击
比如黑名单里有漏网之鱼，不够完整，漏掉了 htaccess 扩展名
那么就有机会，其他情况暂时还没碰到过，希望有经验的朋友可以提出分享一下~

0x05 服务端检测绕过(文件内容检测)

如果文件内容检测设置得比较严格，那么上传攻击将变得非常困难
也可以说它是在代码层检测的最后一道关卡
如果它被突破了，就算没有代码层的漏洞
也给后面利用应用层的解析漏洞带来了机会

我们这里主要以最常见的图像类型内容检测来举例

- 文件幻数检测

主要是检测文件内容开始处的文件幻数，比如图片类型的文件幻数如下

要绕过 jpg 文件幻数检测就要在文件开头写上下图的值

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	00	00	01	ÿØÿà JFIF

Value = FF D8 FF E0 00 10 4A 46 49 46

要绕过 gif 文件幻数检测就要在文件开头写上下图的值

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	47	49	46	38	39	61	0A	00	0A	00	D5	00	00	00	00	00	GIF89a

Value = 47 49 46 38 39 61

要绕过 png 文件幻数检测就要在文件开头写上下面的值

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	‰PNG

Value = 89 50 4E 47

然后在文件幻数后面加上自己的一句话木马代码就行了

- 文件相关信息检测

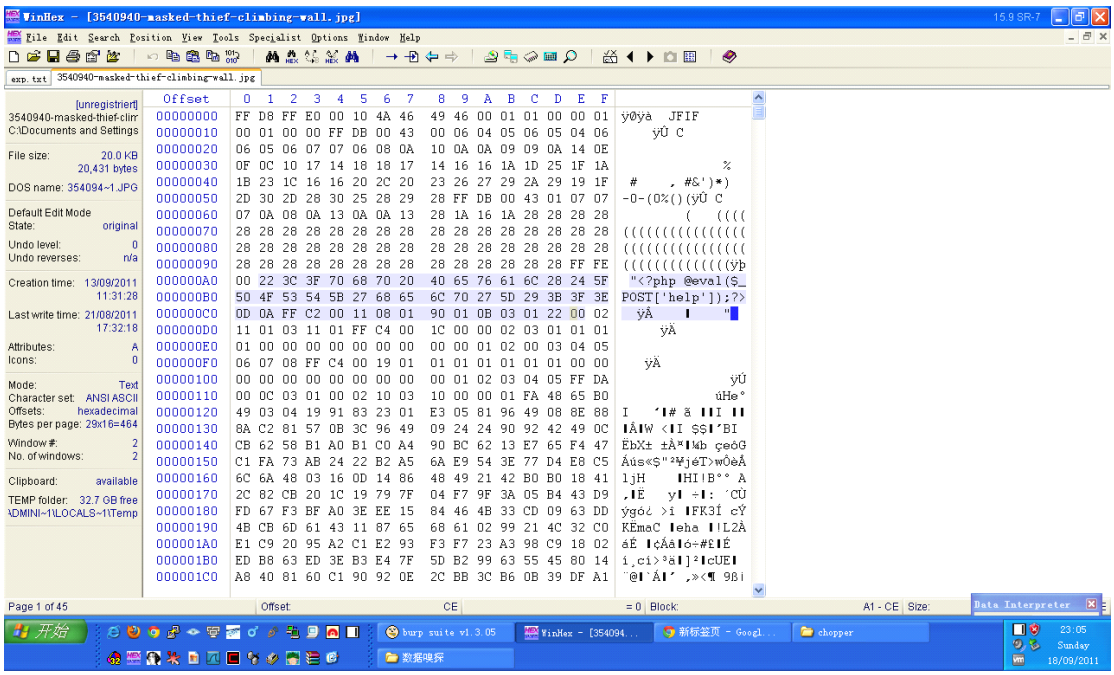
图像文件相关信息检测常用的就是 getimagesize()函数
只需要把文件头部分伪造好就 ok 了，就是在幻数的基础上还加了一些文件信息
有点像下面的结构

GIF89a
(...some binary data for image...)
<?php phpinfo(); ?>
(... skipping the rest of binary data ...)

- 文件加载检测

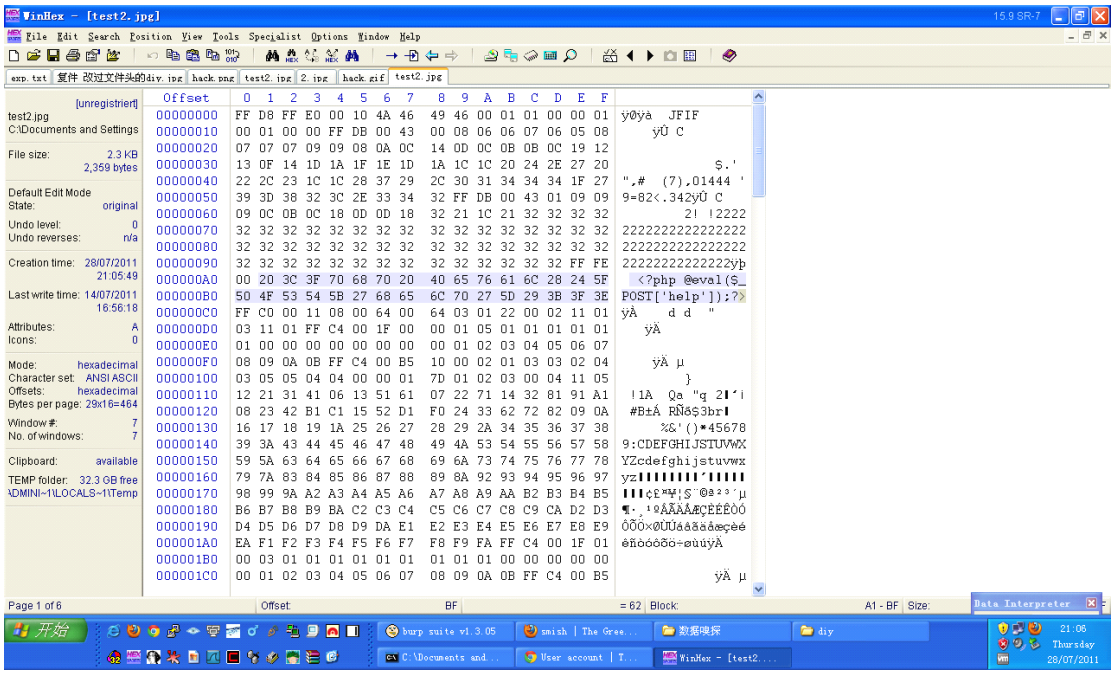
这个是最变态的检测了，一般是调用 API 或函数去进行文件加载测试
常见的是图像渲染测试，再变态点的甚至是进行二次渲染(后面会提到)
对渲染/加载测试的攻击方式是代码注入绕过
对二次渲染的攻击方式是攻击文件加载器自身

先说下对渲染/加载测试攻击 - 代码注入绕过
可以用图像处理软件对一张图片进行代码注入
用 winhex 看数据可以分析出这类工具的原理是
在不破坏文件本身的渲染情况下找一个空白区进行填充代码，一般会是图片的注释区
对于渲染测试基本上都能绕过，毕竟本身的文件结构是完整的

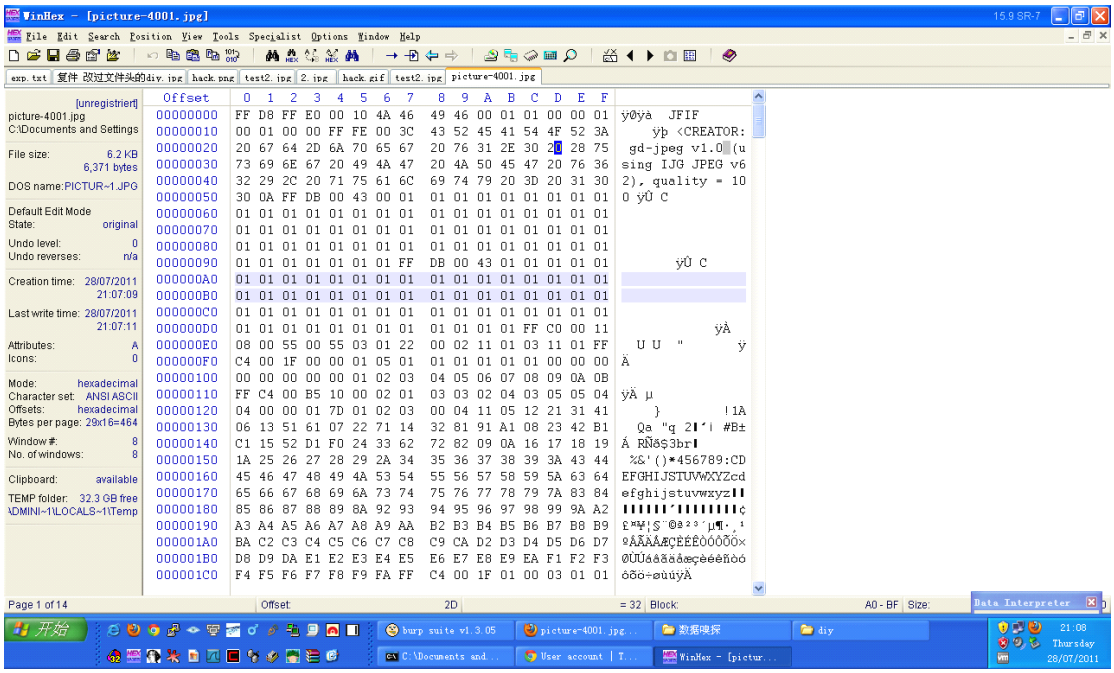


但如果碰到变态的二次渲染
基本上就没法绕过了，估计就只能对文件加载器进行攻击了
下面将来介绍二次渲染

比如上传文件前，文件的数据如下



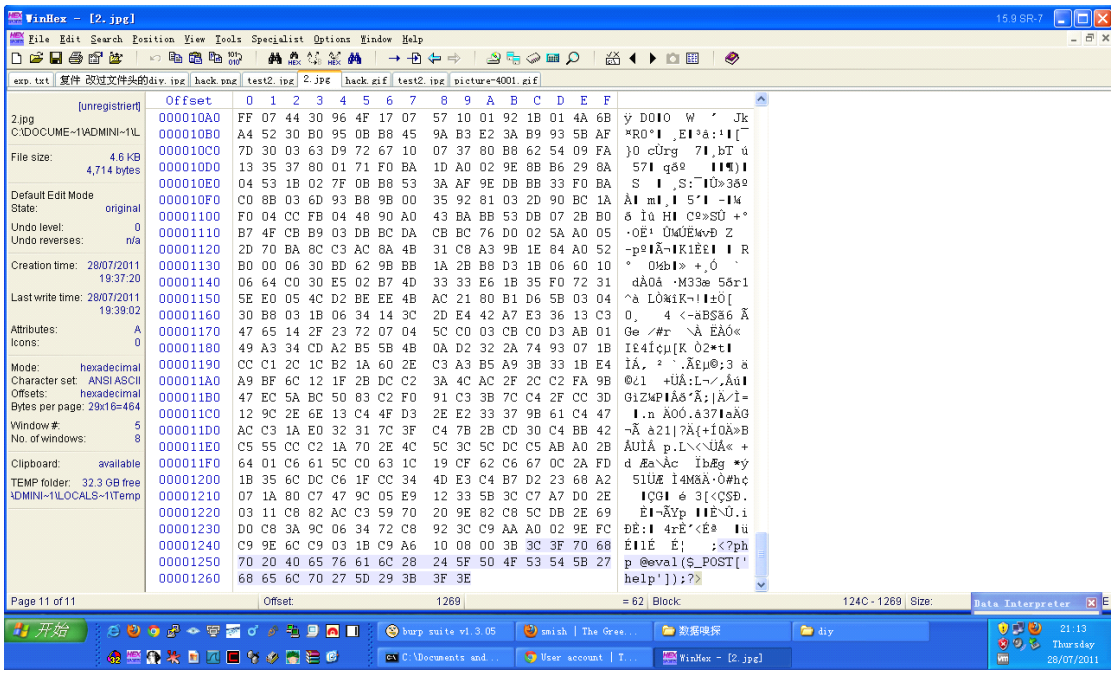
然后上传这个 jpg 但把它重新下载回本地发现了奇怪的地方



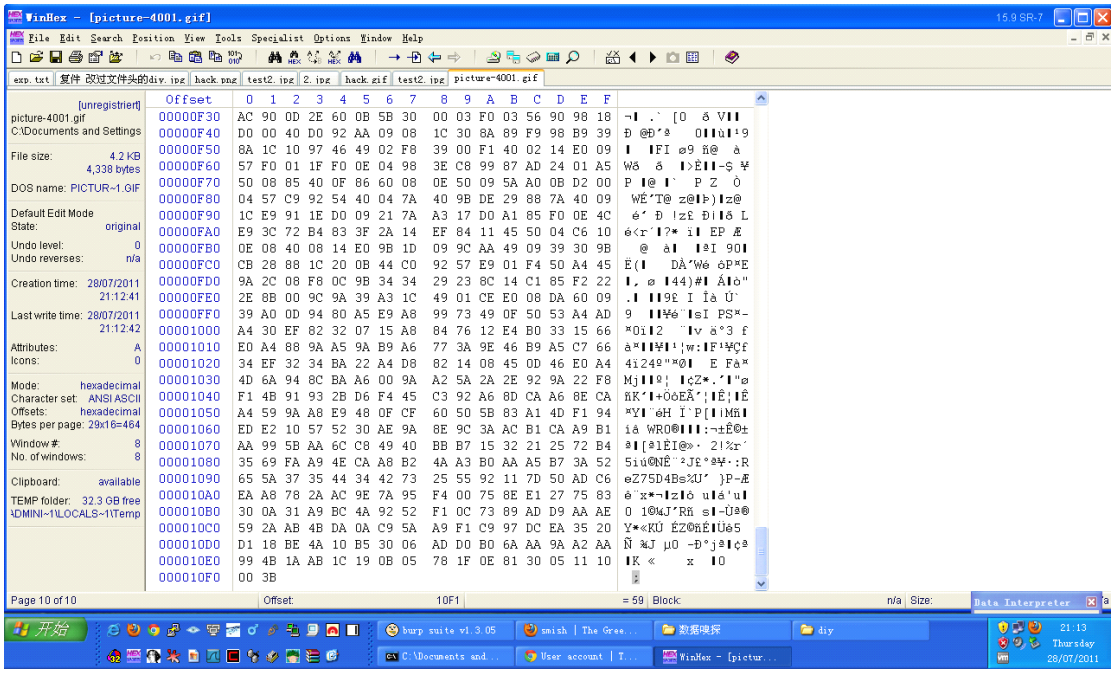
我们的 php 一句话木马不见了

可以看出上传后，图片被二次渲染过
新的 JPG 图片内容里含有这个
CREATOR: gd-jpeg v1.0 (using IJG JPEG v62)
可以看出是调用的 GD php 的 gd 库

测试了 gif 文件也一样
原文件内容是（虽然文件名是 2.jpg，实际文件格式是 gif 哈)



上传后下载回来对比
可以发现文件被重新渲染过
一句话代码也不见了



然后是进行 fuzzing 触发报错 看下是被用什么 API 或函数进行的二次渲染

上传文件数据不完整的 gif 文件
触发报错后，知道后台用的是 imagecreatefromgif()这个函数

上传文件数据不完整的 png 文件
触发报错后，知道后台用的是 imagecreatefrompng()这个函数

某后台调用 GD 库对图像进行二次渲染的代码

```
function image_gd_open($file, $extension)
{
   $extension = str_replace('jpg', 'jpeg', $extension);
   $open_func = 'imageCreateFrom'. $extension;    //函数名变成 imageCreateFrompng 之类
   if (!function_exists($open_func))
   {
      return FALSE;
   }
   return $open_func($file); //变成  imagecreatefrompng('/tmp/php0lbTOn')
}
```

一般进行遇到二次渲染，想绕过，就目前个人经验还没想出方法
它相当于是把原本属于图像数据的部分抓了出来，再用自己的 API 或函数进行重新渲染
在这个过程中非图像数据的部分直接就被隔离开了

能想到的一个思路就是基于数据二义性，即让数据既是图像数据也包含一句话木马代码
就像 shellcode 通过数据二义性绕过 IDS 检测特殊字符一样的道理
但现在我还不知道怎么构造出这样的图像文件

如果要对文件加载器进行攻击，常见的就是溢出攻击
上传自己的恶意文件后，服务器上的文件加载器会主动进行加载测试
加载测试时被溢出攻击执行 shellcode
比如 access/mdb 溢出，大家可以参考下 http://lcx.cc/?FoxNews=1542.html

总之对文件完整性检测的绕过，通常就直接用个结构完整的文件进行代码注入即可
没必要再去测到底是检查的幻数还是文件头结构之类的了

0x06 解析攻击

我们以一个故事的方式来描述，方便大家能更深层次地理解一些渗透的本质
古时候，A 国对 B 国进行攻城战，B 国背后有一条水道，这时候会有以下情况
[*] 完全没防御
[*] 有一点简单的防御 (用木头制成做成的门栏)
[*] 有一定程度的防御 (用金属制成做成的门栏)
[*] 非常强的防御 (几乎完全封闭死了)

从攻击角度来分是这样的
1. 直接解析 (完全没防御或有一点简单的防御)
比如我们直接就可以上传一个扩展名是.php 的文件
只需要简单地绕过客户端 javascript 检测或者服务端 MIME 类型检测就行了

这样来描述一下
A 国直接派了一队精英小分队从水道里直接破坏掉一些普通的木头门栏之类
攻入了 B 国内部

2. 配合解析 (有一定程度的防御)
我们可以理解为先代码注入到服务器上，上传一个带有一句话木马的图片或文件之类
让它待在某个位置，等待这一个解析的配合
比如 php 的文件包含解析，web 服务器的解析漏洞，.htaccess 解析等

这样来描述下
A 国先派了一队精英小分队潜入了 B 国的水道，但是有铁门锁着，没法直接打开
比如要等待一个 B 国的间谍来接应他们，把铁门打开，他们才能进行下一步动作
随着安全意识的增强，现在基本上的主流攻击都在配合解析这一块
所以了解各类解析漏洞/机制是相当重要的
下面会更细致地为解析攻击进行分类

- 网络渗透的本质

这部分大家纯当 YY 看来玩就是了，也不用对此争论什么，我也并不是在对网络渗透做一个以偏概全的总结，只是抽象了其中某一部分来看。

其实绝大部分网络渗透的本质(技术部分)就如上面总结的情况
主体就是代码注入+代码解析/执行
这种模式贯穿了几乎主流的技术性渗透攻击
像缓冲区溢出攻击，sql 注入攻击，文件上传攻击，文件包含攻击，脚本代码注入等等

主要是两类情况

1. 直接解析/执行攻击

像缓冲区溢出和 sql 注入攻击，脚本代码注入就是很明显的属于这里攻击
直接将代码注入到一个解析/执行环境里，直接就能让代码得到执行
所以危害性也来得最大，效果最明显
shellcode 注入程序后，直接劫持 EIP，进行该系统环境权限做任何操作
sql 命令注入数据库后，直接就能执行该数据库账号权限下的任何操作

2. 配合解析/执行攻击

算是一种组合攻击，在这类情况下
往往不像第一种情况能拥有直接的解析/执行环境
比较明显的就是我们的上传攻击
我们需要先上传数据(注入代码)到服务端上去
然后想办法去调用解析/执行环境(比如 Web 应用程序解析漏洞)
来解析/执行已经注入到了服务端的代码

如果对渗透的技术部分(社工更多属于非技术因素)看得更本质的话
就是上面描述的代码注入+解析/执行代码
把很多看上去很宽泛、很繁琐的事物
提取出它们的共性，就能更有针对性地对付一类问题

YY 完了网络渗透，接下来开始正式的部分
我们从解析攻击的具体方式来分类如下

- 直接解析

能以 asp,php 之类的扩展名存储在服务器上

- 本地文件包含解析

主要是 php 的本地文件包含 (远程文件包含不属于上传攻击绕过范畴)

- .htaccess 解析

就不用多说了，看看之前.htaccess 文件攻击的那个案例
用户自己定义如何去调用解析器解析文件就可以了

- web 应用程序解析漏洞以及其原理

Apache/IIS/Nginx 解析漏洞
下面将重点来总结各类 web 应用程序解析漏洞
Apache/IIS/Nginx 的各版本基本上都收集到了(包括老版本)
但出于时间原因，Apache 的没时间测单独的版本
都是测的集成环境，如常见的 WampServer，AppServ
然后 IIS 的版本测试完毕
Nginx 的也因为没有时间测单独的版本，但一般漏洞网站都给出版本范围
会先以环境作分类，这样更适合于从渗透人员的角度
然后会再以漏洞原理作分类，这样更能清楚一些本质的东西

[*] Apache 解析漏洞

解析 - test.php.任意不属于黑名单且也不属于 Apache 解析白名单的名称
描述 - 一个文件名为 x1.x2.x3 的文件，Apache 会从 x3 的位置往 x1 的位置开始尝试解析
如果 x3 不属于 Apache 能解析的扩展名，那么 Apache 会尝试去解析 x2 的位置，
这样一直往前尝试，直到遇到一个能解析的扩展名为止
测试 - 测试了下面这些集成环境，都以它们的最新版本来测试，应该能覆盖所有低版本

WampServer2.0 All Version (WampServer2.0i / Apache 2.2.11)	[Success]
WampServer2.1 All Version (WampServer2.1e-x32 / Apache 2.2.17)	[Success]
Wamp5 All Version (Wamp5_1.7.4 / Apache 2.2.6)	[Success]
AppServ 2.4 All Version (AppServ - 2.4.9 / Apache 2.0.59)	[Success]
AppServ 2.5 All Version (AppServ - 2.5.10 / Apache 2.2.8)	[Success]
AppServ 2.6 All Version (AppServ - 2.6.0 / Apache 2.2.8)	[Success]

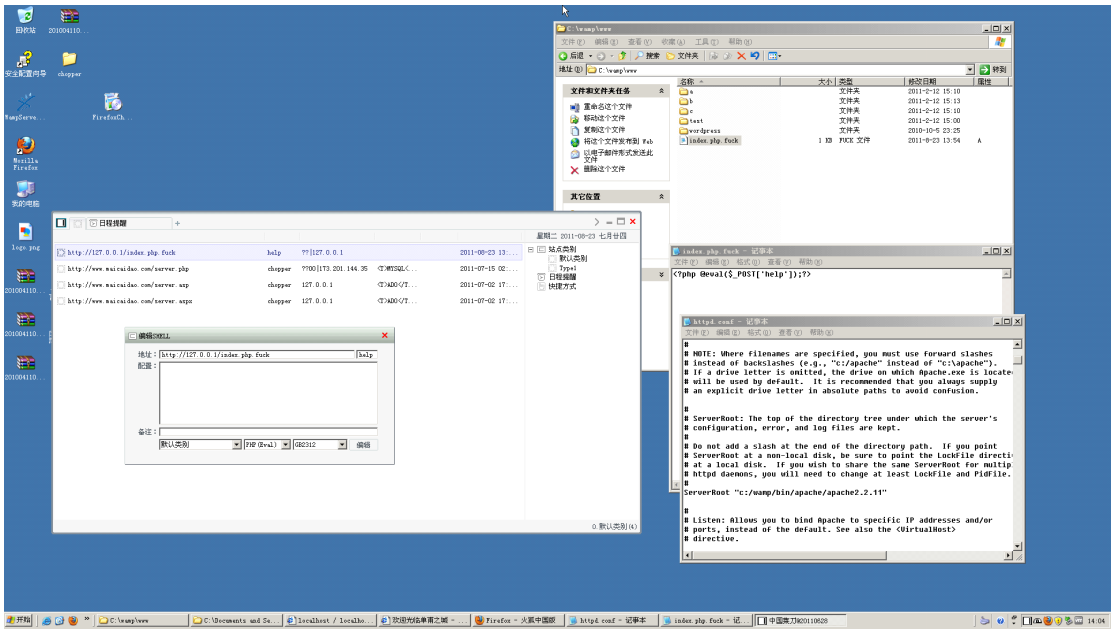
上面测试过的集成环境都有这个扩展名解析顺序漏洞，然后所有测试过的集成环境都有对 php3 扩展名按 php 解析这个小洞 (本质上来说这个不算漏洞，只是在针对一些名单不全的黑名单时，能有绕过的机会)。

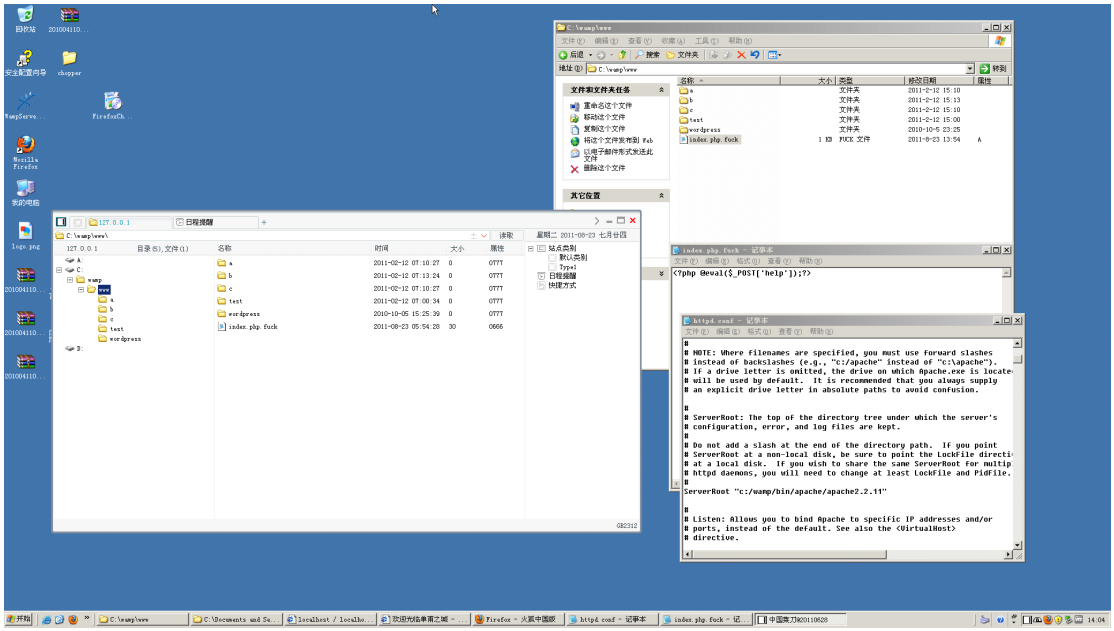
个人是觉得像这些解析漏洞市面上的都只是说有这么一个漏洞，都没怎么给出过详细的版本范围之类，而渗透不是猜测，最好要做到心知肚明，有矢放的。具体这个解析漏洞到底波及 Apache 哪些版本和范围，目前我也没时间挨个测试，只有等我有时间了，测出来了再附带上来。

其实这个解析漏洞大家可以根据上面一些集成环境的 Apache 的版本大概预测下，至少还是有一定几率的，比如 Apache 2.0.x <= 2.0.59，Apache 2.2.x <= 2.2.17，Apache 2.2.2 <= 2.2.8 之类，当然这些只是一个从高版本对低版本的预测，实际效果还是要测试后才知道，我只是在这里提一下而已，也不失为一种思路。

WampServer2.0 All Version (Apache 2.2.11)

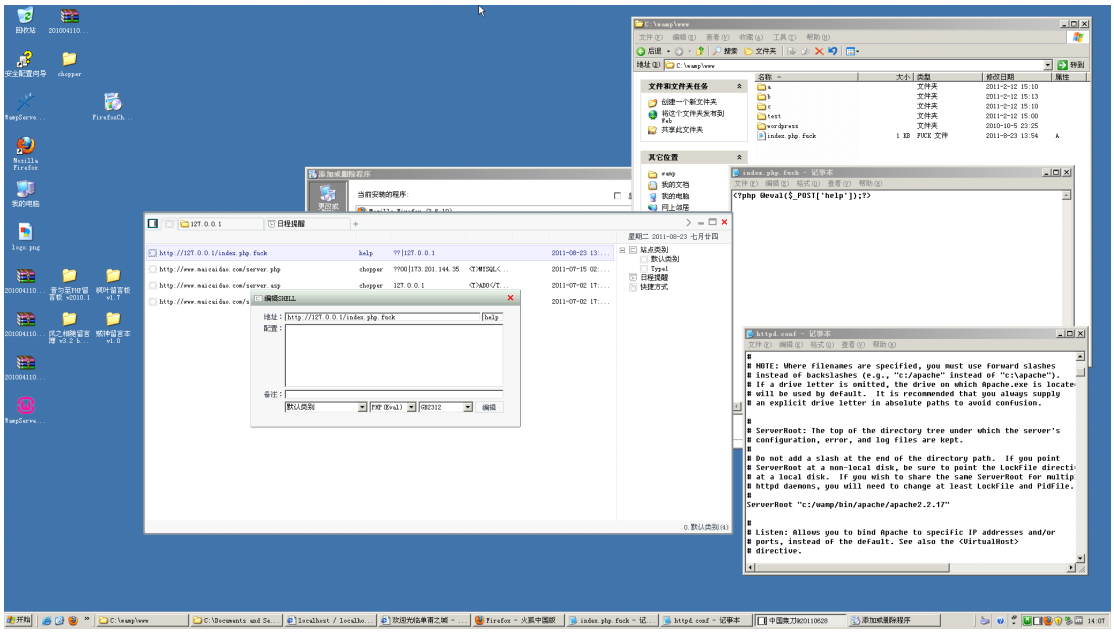
以 WampServer2.0 最新版 WampServer2.0i 版来测试的

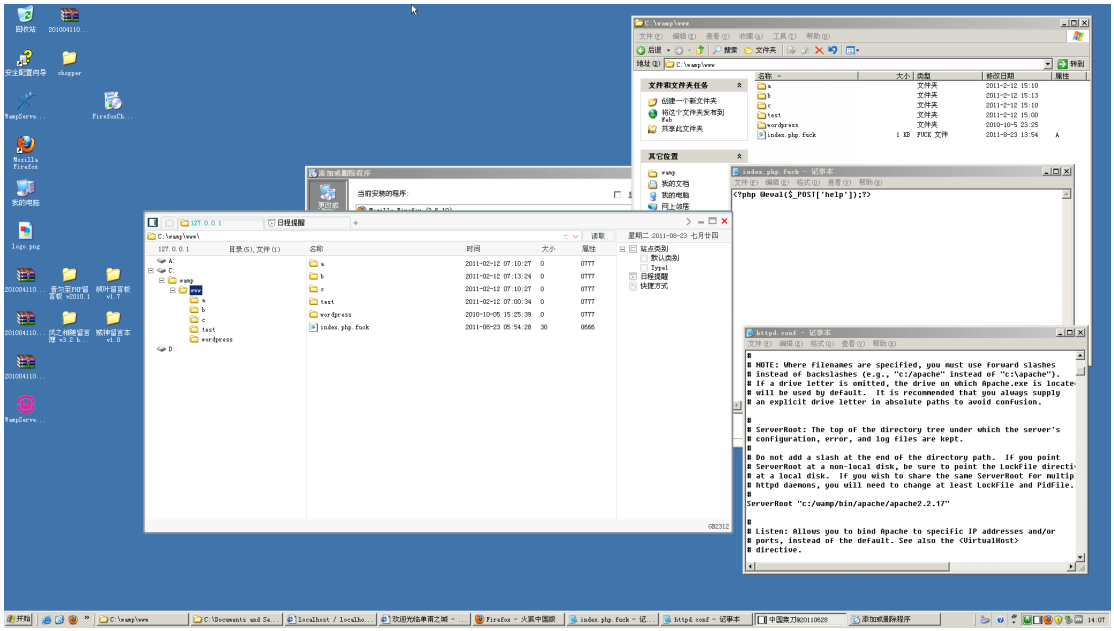




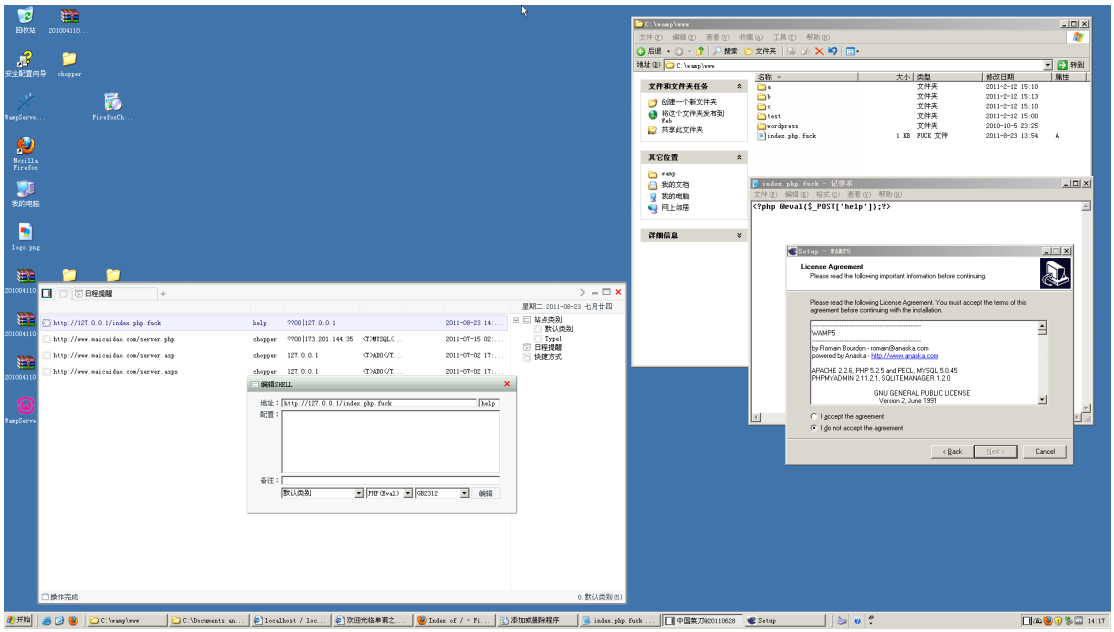
WampServer2.1 All Version (Apache 2.2.17)

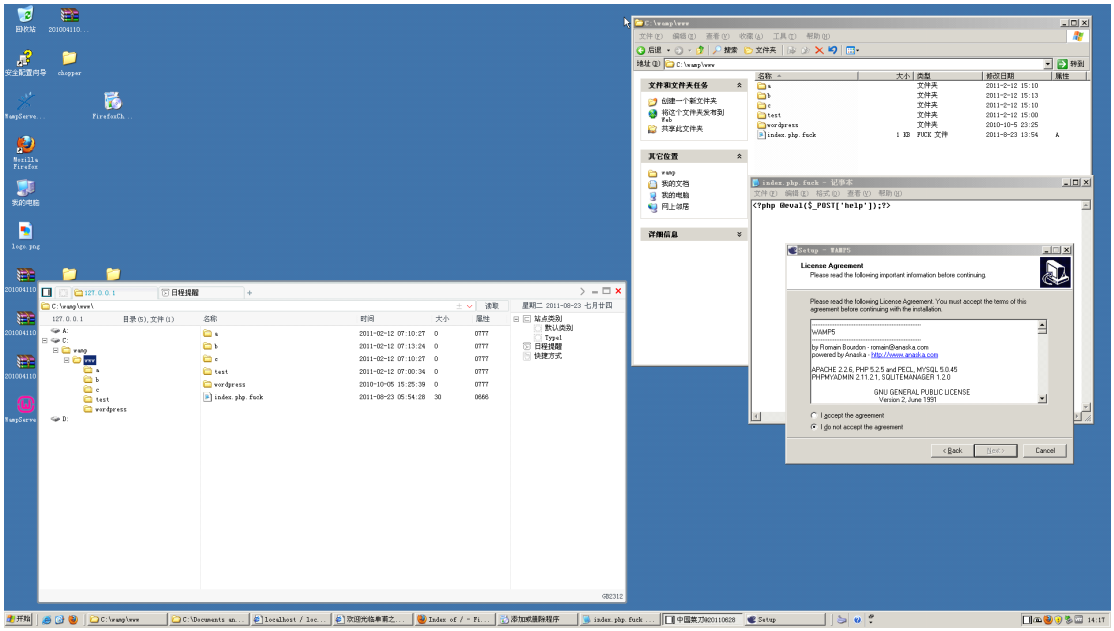
以 WampServer2.1 最新版 WampServer2.1e-x32 版来测试的



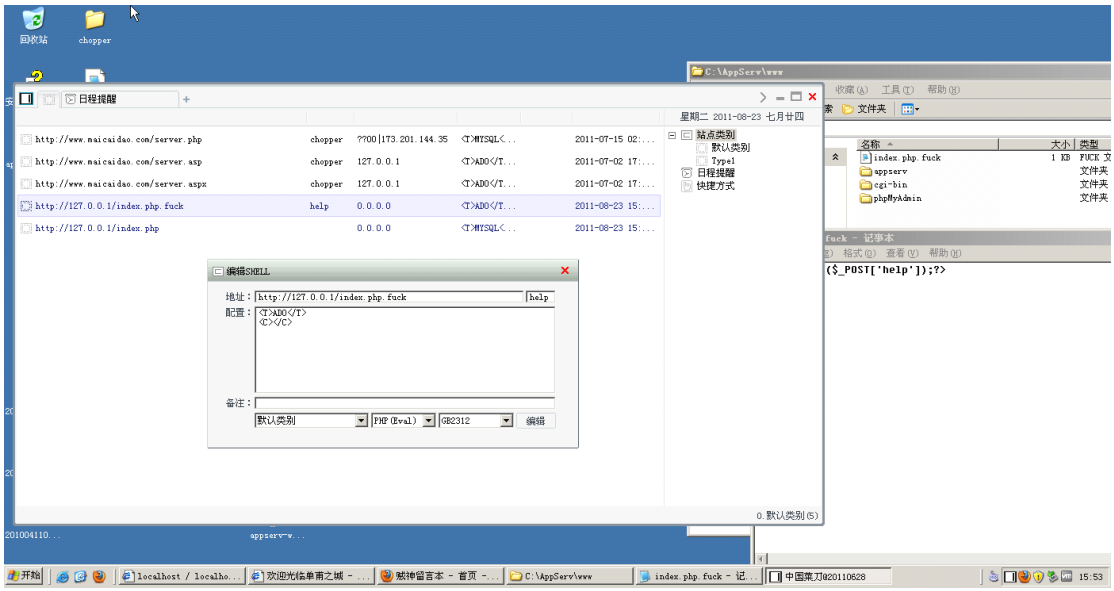


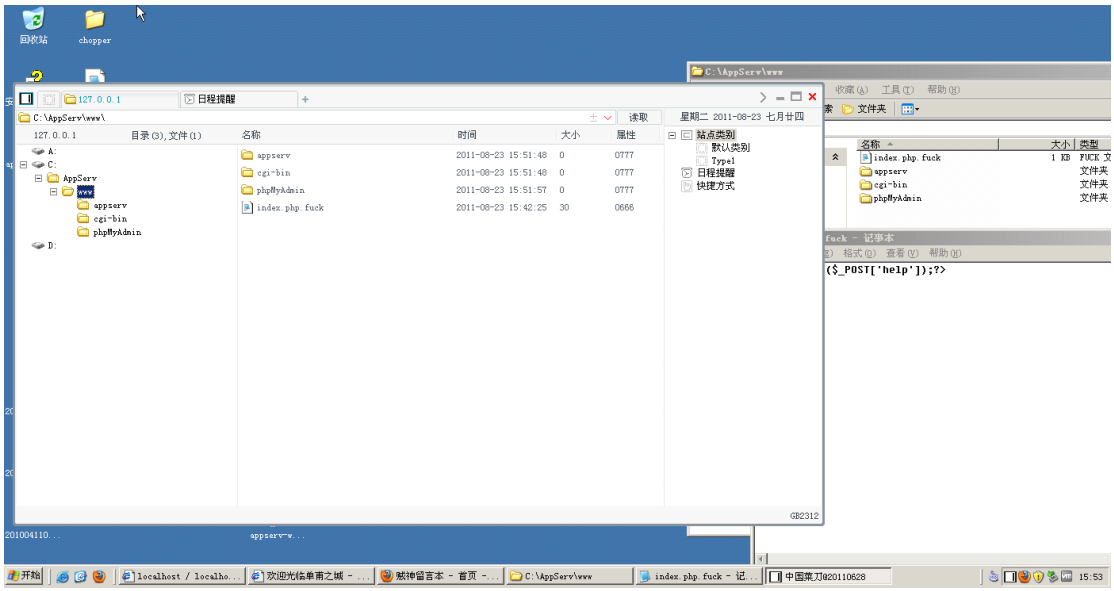
Wamp5 All Version (Apache 2.2.6)
以 Wamp5 最新版 Wamp5_1.7.4 版来测试的



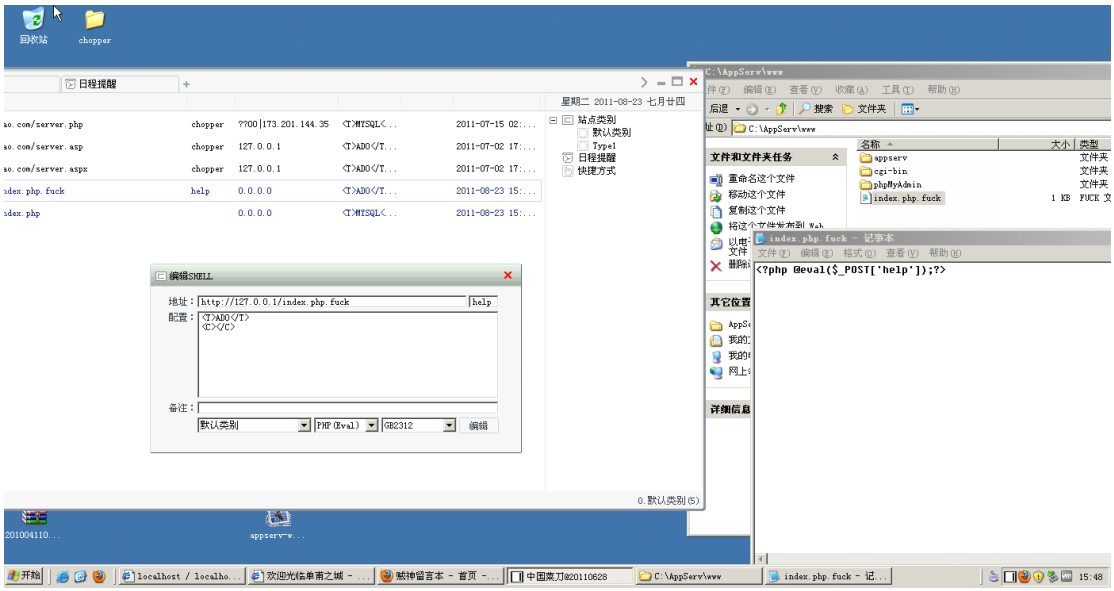


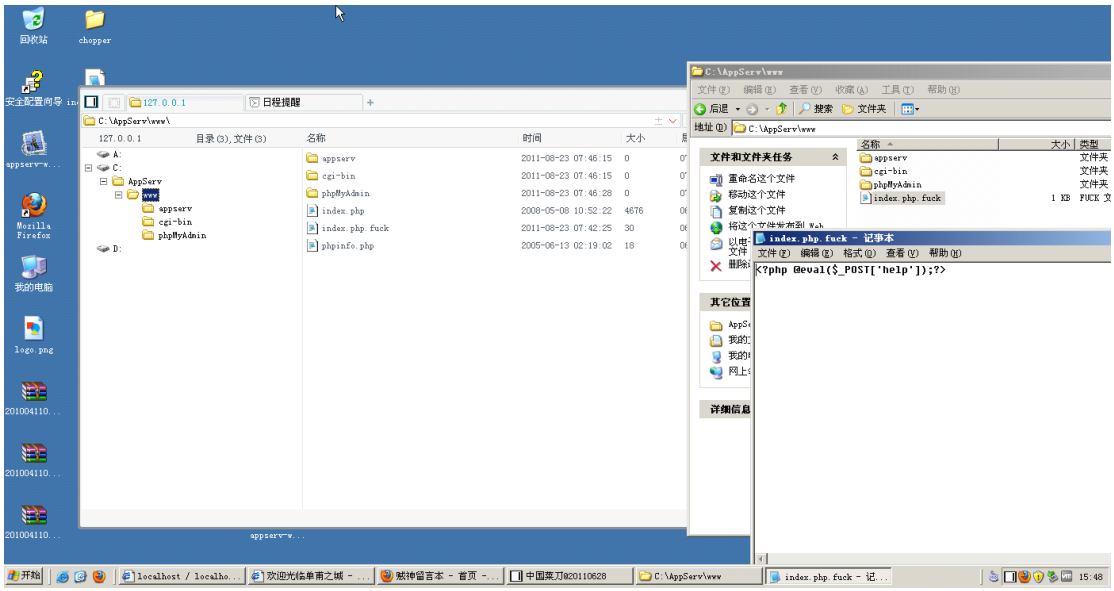
AppServ 2.4 All Version (Apache 2.0.59)
以 AppServ 2.4 最新版 AppServ - 2.4.9 版来测试的



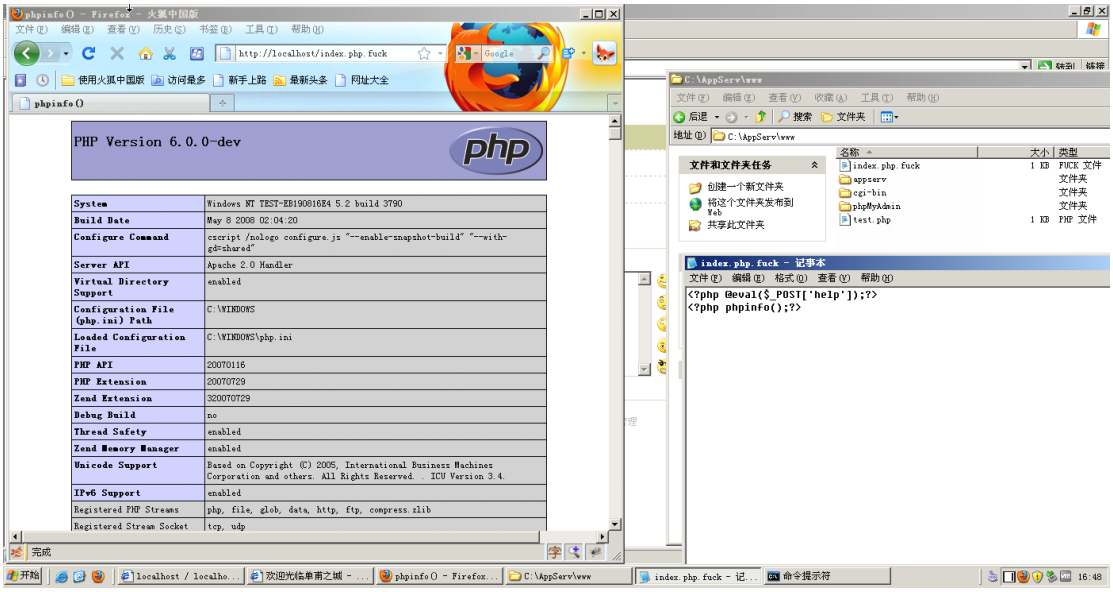


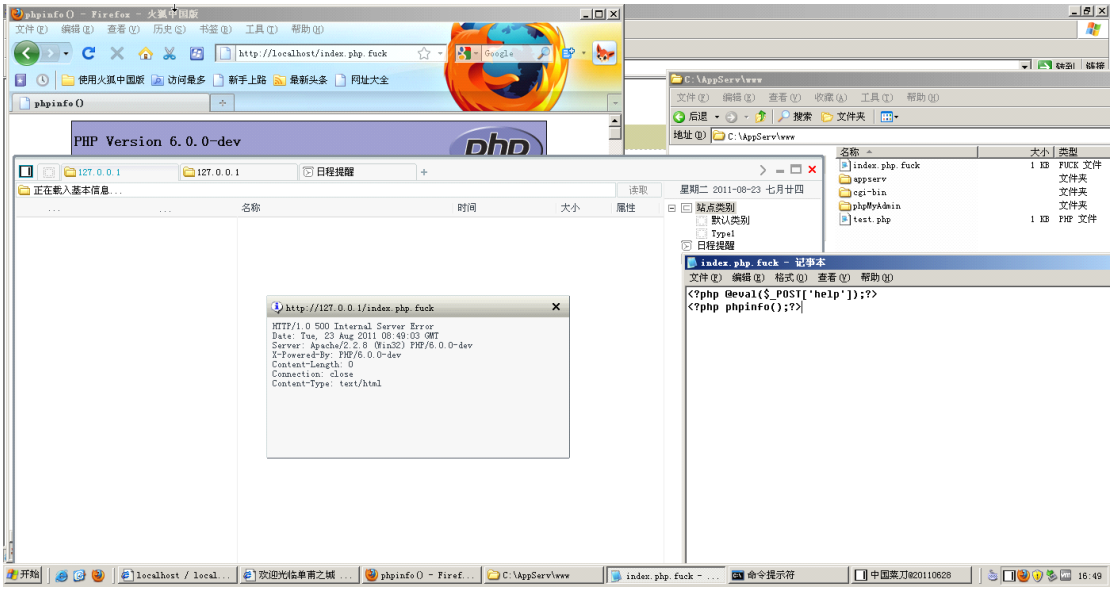
AppServ 2.5 All Version (Apache 2.2.8)
以 AppServ 2.5 最新版 AppServ - 2.5.10 版来测试的





AppServ 2.6 All Version (Apache 2.2.8)
以 AppServ 2.6 最新版 AppServ - 2.6.0 版来测试的
用中国菜刀进行一句话连接时不知道为什么失败了，但是用 phpinfo()测试了还是可以解析的





[*] IIS 解析漏洞

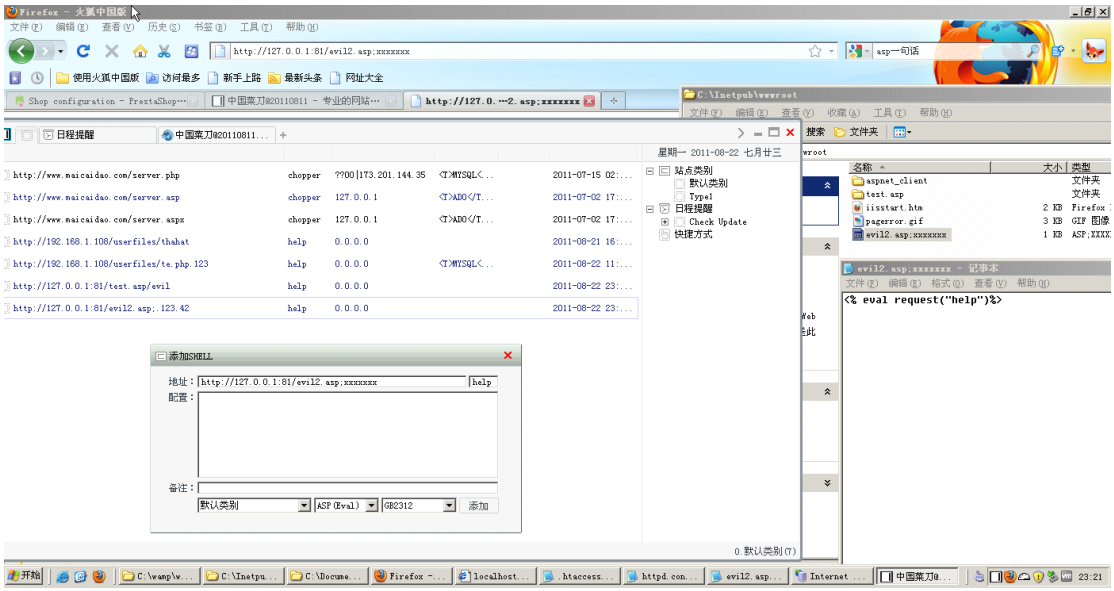
解析 - test.asp/任意文件名 | test.asp;任意文件名 | 任意文件名/任意文件名.php
描述 - IIS6.0 在解析 asp 格式的时候有两个解析漏洞，一个是如果目录名包含".asp"字符串，那么这个目录下所有的文件都会按照 asp 去解析，另一个是只要文件名中含有".asp;"会优先按 asp 来解析
IIS7.0/7.5 是对 php 解析时有一个类似于 Nginx 的解析漏洞，对任意文件名只要在 URL 后面追加上字符串"/任意文件名.php"就会按照 php 的方式去解析(IIS6.0 没测试)
测试 - 测试了下面这些集成环境，都以它们的最新版本来测试，应该能覆盖所有低版本

IIS6.0 (Win2003 SP2 + IIS6.0)	[Success]
IIS7.0 (Win2008 R1 + IIS7.0)	[Success]
IIS7.5 (Win2008 R2 + IIS7.5)	[Success]

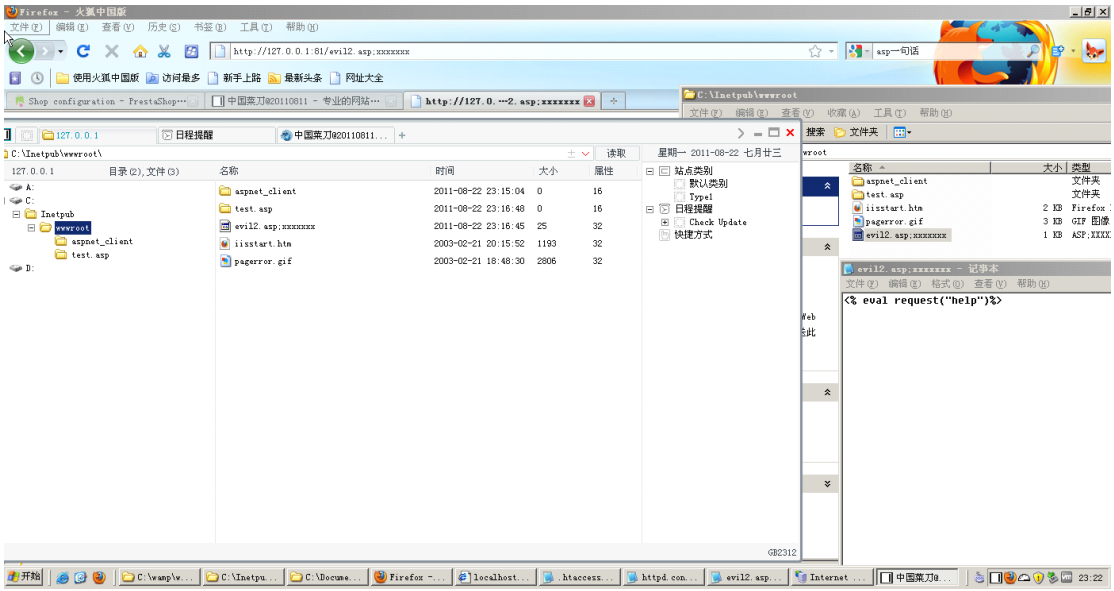
IIS 的解析漏洞在市面上描述的版本还算明确，不像 Apache 那么模糊，针对 IIS6.0，只要文件名不被重命名基本都能搞定。这里要注意一点，对于任意文件名/任意文件名.php 这个漏洞其实是出现自 php-cgi 的漏洞，所以其实跟 IIS 自身是无关的，这个会在接下来讲到。

IIS6.0 (asp 解析漏洞 1)

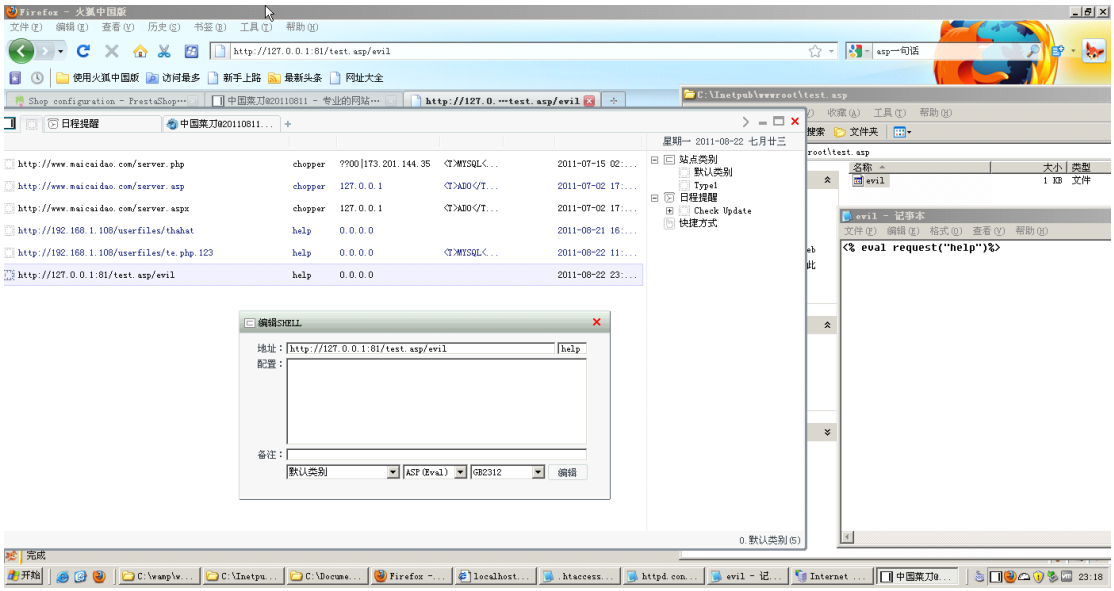
以 win2003 sp2 + IIS6.0 来测试



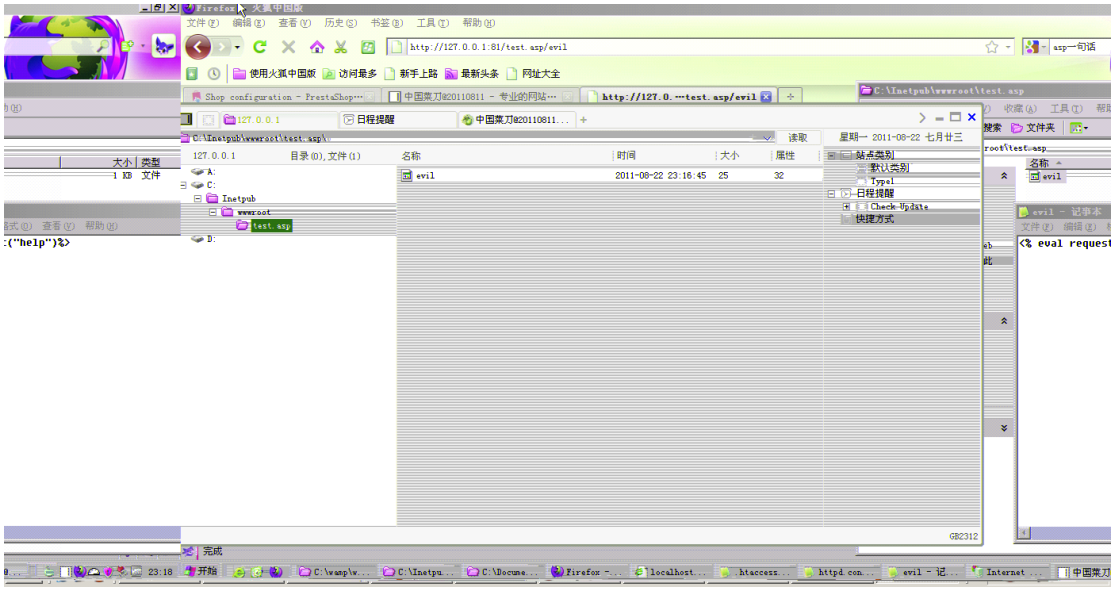
图中的文件命名为 evil2.asp;xxxxxxx



IIS6.0 (asp 解析漏洞 2)
以 win2003 sp2 + IIS6.0 来测试

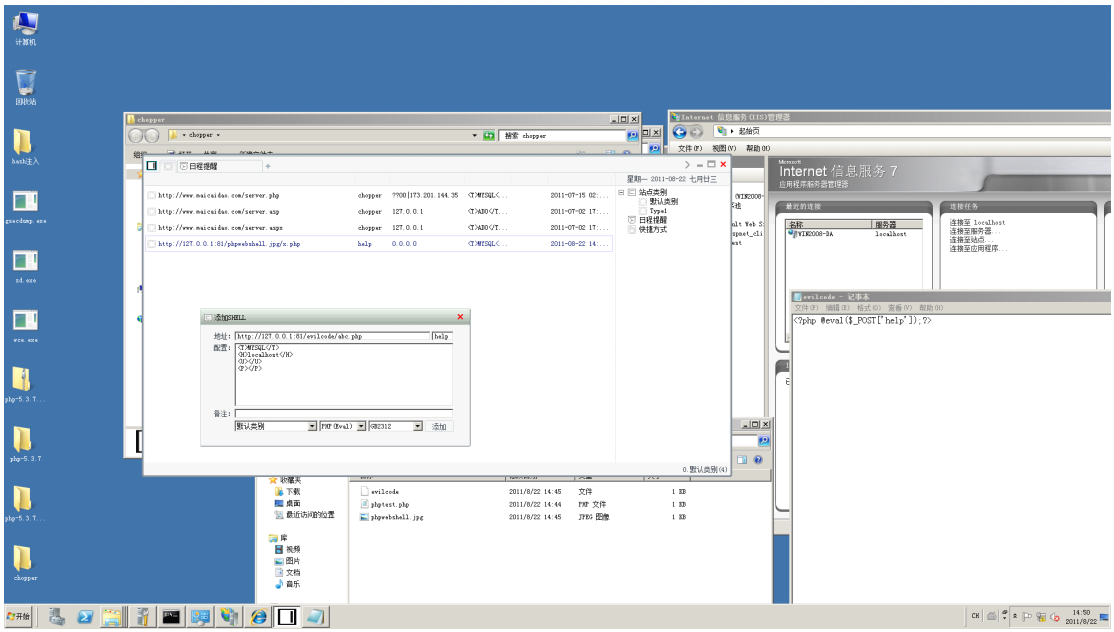


图中的文件命名为 evil
并位于目录 test.asp 下

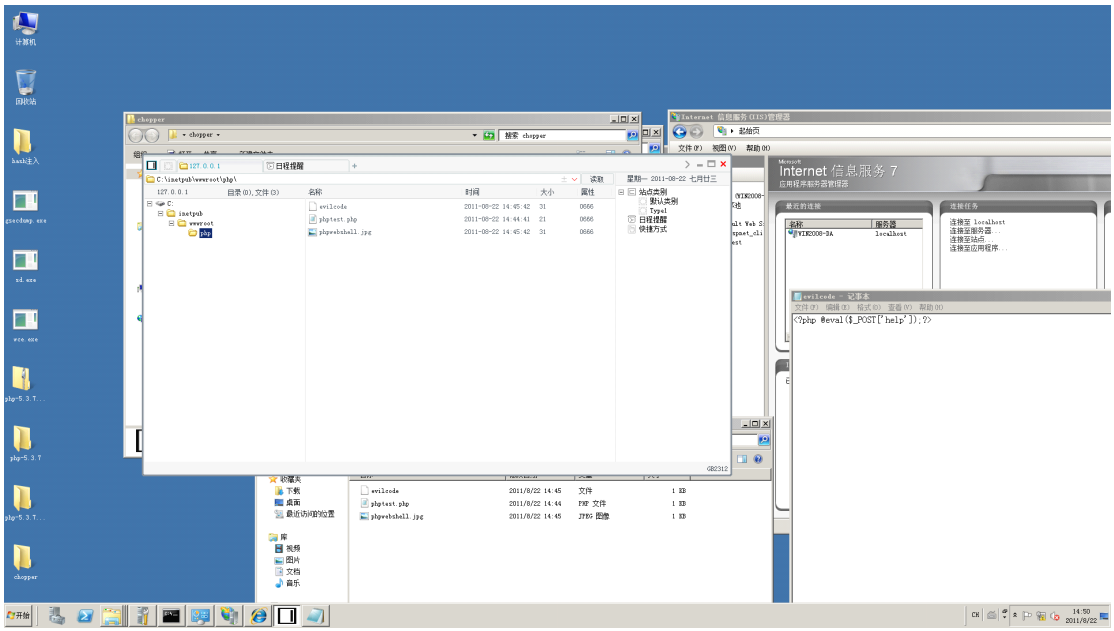


IIS7.5 (php 解析漏洞)

以 win2008 r2 + IIS7.5 来测试



对 evilcode 这个文件进行 php 解析漏洞测试



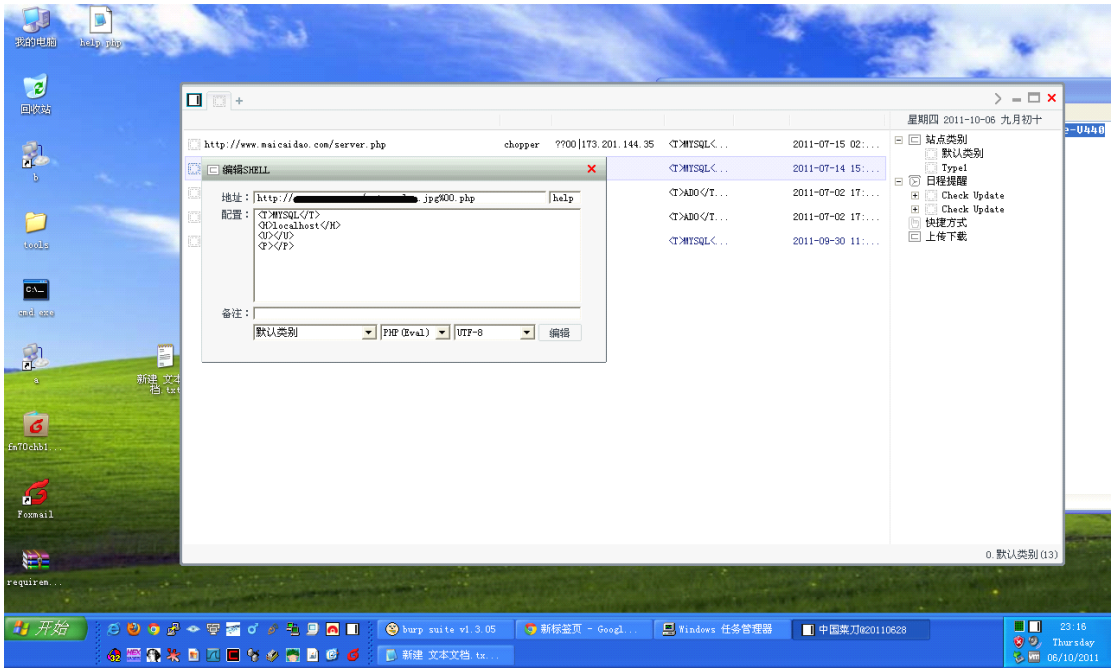
[*] Nginx 解析漏洞

解析 - 任意文件名/任意文件名.php | 任意文件名%00.php
描述 - 目前 Nginx 主要有这两种漏洞，一个是对任意文件名，在后面添加/任意文件名.php 的解析漏洞，比如原本文件名是 test.jpg，可以添加为 test.jpg/x.php 进行解析攻击。还有一种是对低版本的 Nginx 可以在任意文件名后面添加%00.php 进行解析攻击。
测试 - 测试了下面这些环境

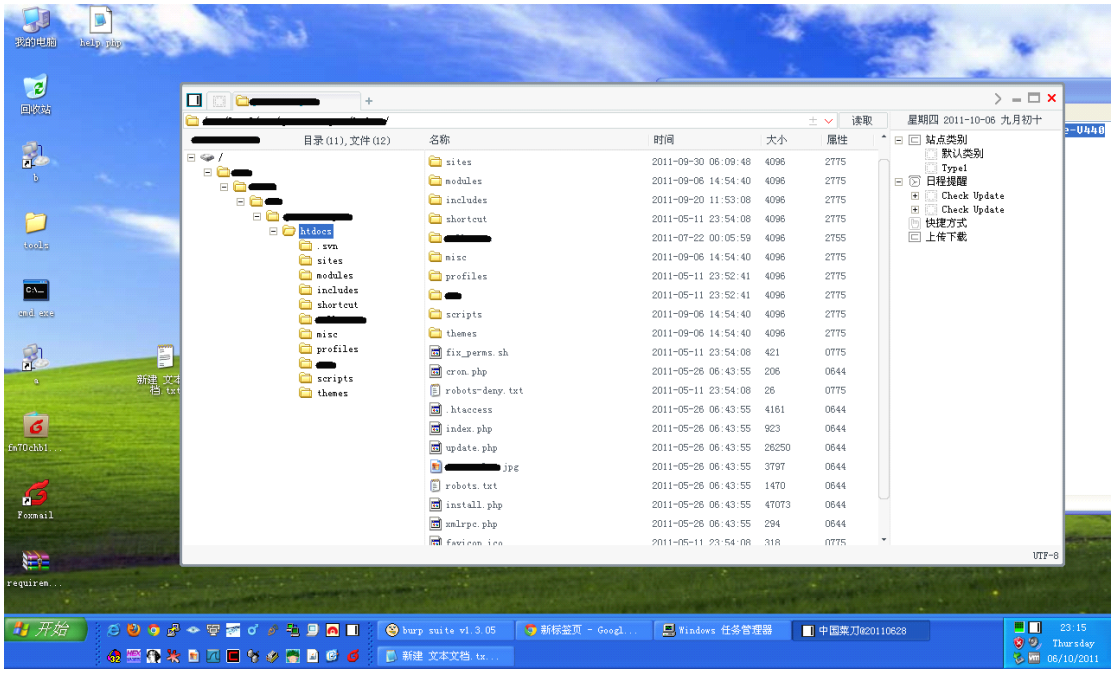
nginx 0.5.*	[Success]
nginx 0.6.*	[Success]
nginx 0.7 <= 0.7.65	[Success]
nginx 0.8 <= 0.8.37	[Success]

这里要注意一点，对于任意文件名/任意文件名.php 这个漏洞其实是出现自 php-cgi 的漏洞，所以其实跟 nginx 自身是无关的，这个会在接下来讲到。

Nginx/0.6.32 (php 解析漏洞 2)
以 Linux + Nginx/0.6.32 来测试



上传图片，然后通过 xx.jpg%00.php 解析漏洞连接一句话木马



从漏洞原理来说总归有 4 类 web 应用程序解析漏洞
[*] Apache 的扩展名顺序解析漏洞
这个是 Apache 自身的漏洞

[*] IIS 的 asp 解析漏洞
这个是 IIS 自身的漏洞

[*] Nginx 的%00 解析漏洞
这个是 Nginx 自身的漏洞

[*] php-cgi 的默认配置漏洞
这类漏洞主要出现在 IIS 和 Nginx 这类以 CGI 形式调用 php 的 web 应用程序
而 Apache 通常是以 module 的形式去调用 php，所以很少出现这个漏洞

Bug #50852 FastCGI Responder's accept_path_info behavior needs to be optional
地址: https://bugs.php.net/bug.php?id=50852&edit=1

这个漏洞由 cgi.fix_pathinfo 的值造成的

```
; cgi.fix_pathinfo provides *real* PATH_INFO/PATH_TRANSLATED support for CGI.  PHP's
; previous behaviour was to set PATH_TRANSLATED to SCRIPT_FILENAME, and to not grok
; what PATH_INFO is.  For more information on PATH_INFO, see the cgi specs.  Setting
; this to 1 will cause PHP CGI to fix its paths to conform to the spec.  A setting
; of zero causes PHP to behave as before.  Default is 1.  You should fix your scripts
; to use SCRIPT_FILENAME rather than PATH_TRANSLATED.
; http://php.net/cgi.fix-pathinfo
;cgi.fix_pathinfo=1
```

php-cgi 默认的配置里 cgi.fix_pathinfo 是被注释掉的
下图是再 cgi.fix_pathinfo 是被注释掉的情况下 phpinfo()里的信息

cgi-fcgi

Directive	Local Value	Master Value
cgi.check_shebang_line	1	1
cgi.discard_path	0	0
cgi.fix_pathinfo	1	1
cgi.force_redirect	1	1
cgi.nph	0	0
cgi.redirect_status_env	*no value*	*no value*
cgi.rfc2616_headers	0	0
fastcgi.impersonate	0	0

Internet | 保护模式: 禁用

```
; cgi.fix_pathinfo provides *real* PATH_INFO/PATH_TRANSLATED support for CGI.  PHP's
; previous behaviour was to set PATH_TRANSLATED to SCRIPT_FILENAME, and to not grok
; what PATH_INFO is.  For more information on PATH_INFO, see the cgi specs.  Setting
; this to 1 will cause PHP CGI to fix its paths to conform to the spec.  A setting
; of zero causes PHP to behave as before.  Default is 1.  You should fix your scripts
; to use SCRIPT_FILENAME rather than PATH_TRANSLATED.
; http://php.net/cgi.fix-pathinfo
;cgi.fix_pathinfo=1
```

可以看到这里实际运行效果是以 cgi.fix_pathinfo = 1 在运行
而通常安全意识不高的管理员在安装 IIS+php 或 Nginx+php 的时候都是以默认配置在安装
自然这种情况下，这类服务器全都会产生漏洞

0x07 上传攻击框架

之前的都是从服务端的角度在给上传情况分类
现在我们要从攻击者的角度来给上传情况分类
这是这套 framework 的核心部分
知道了从攻击者的角度如何分类
就等于知道如何来综合分析一套源码
并从中知道是否有存在漏洞的可能性

- 轻量级检测绕过攻击

[*] 绕过 javascript 对扩展名的检测
<用 burp 之类的反向代理工具直接 POST 数据包到服务端，绕过前端检测>

[*] 绕过服务端对 http request 包 MIME 类型检测
<用 burp 之类的反向代理工具伪造 POST 数据包到服务端，绕过 MIME 检测>

- 路径/扩展名检测绕过攻击

[*] 黑名单绕过
文件名大小写绕过
名单列表绕过
特殊文件名绕过
0x00 截断绕过
.htaccess 文件攻击
本地包含漏洞
Apache 解析漏洞
IIS 解析漏洞
Nginx 解析漏洞

[*] 白名单绕过
0x00 截断绕过
本地文件包含漏洞
IIS 解析漏洞
Nginx 解析漏洞

- 文件内容检测绕过攻击

[*] 文件加载测试绕过
<对文件进行代码注入再配合任意解析调用/漏洞>

- 上传攻击框架

然后再直观点，看它们属于什么层面的漏洞

- 轻量级检测绕过攻击

[*] 绕过 javascript 对扩展名的检测 (代码层漏洞)

[*] 绕过服务端对 http request 包 MIME 类型检测 (代码层漏洞)

- 路径/扩展名检测绕过攻击

[*] 黑名单绕过

文件名大小写绕过 (代码层漏洞)

名单列表绕过 (代码层漏洞)

特殊文件名绕过 (代码层漏洞)

0x00 截断绕过 (代码层漏洞)

.htaccess 文件攻击 (代码层漏洞)

php 文件包含漏洞 (代码层漏洞)

Apache 解析漏洞 (应用层漏洞)

IIS 解析漏洞 (应用层漏洞)

Nginx 解析漏洞 (应用层漏洞)

[*] 白名单绕过

0x00 截断绕过 (代码层漏洞)

php 文件包含漏洞 (代码层漏洞)

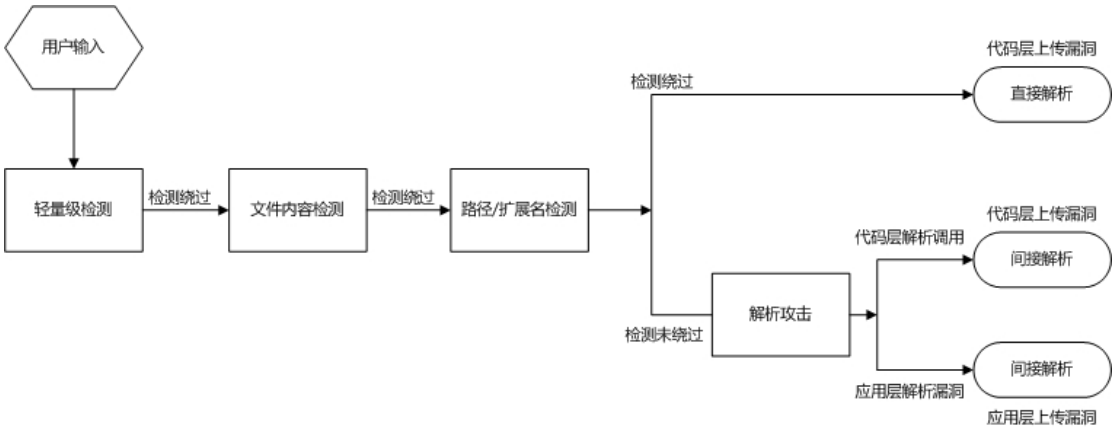
IIS 解析漏洞 (应用层漏洞)

Nginx 解析漏洞 (应用层漏洞)

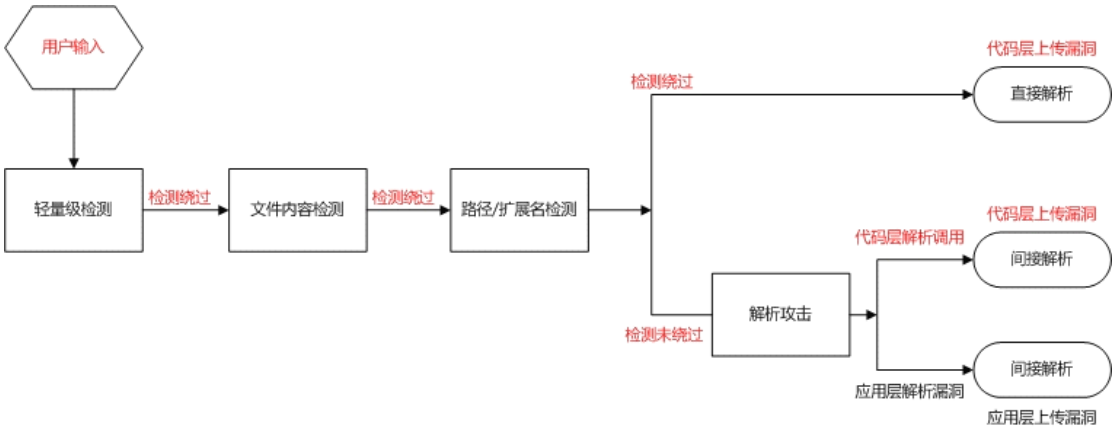
- 文件内容检测绕过攻击

[*] 文件加载绕过 (代码层漏洞)

上传攻击流程如下图



代码层上传漏洞如下图 (红色字体标记部分)

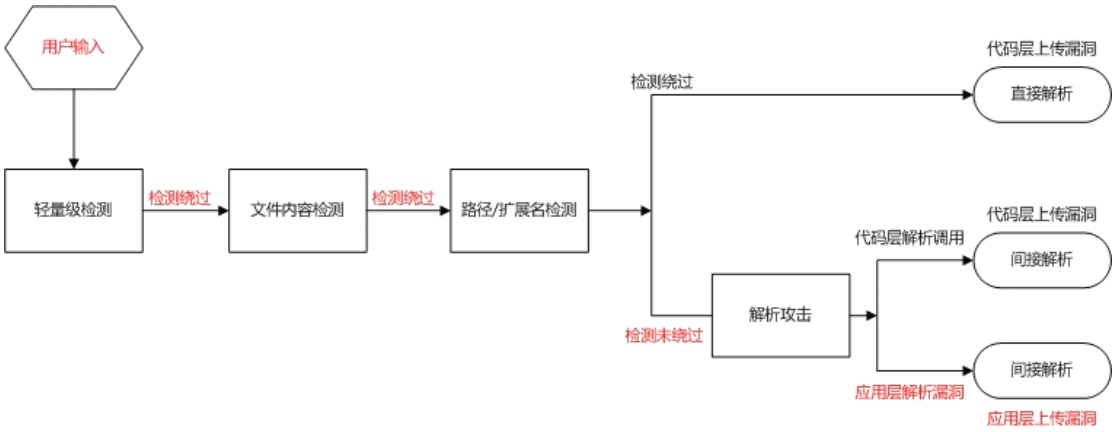


攻击要点:

要绕过轻量级检测
要绕过文件内容检测

A. 要绕过代码层对路径/扩展名检测 (直接解析)
B. 找到代码层的解析调用 (间接解析)

应用层上传漏洞如下图 (红色字体标记部分)

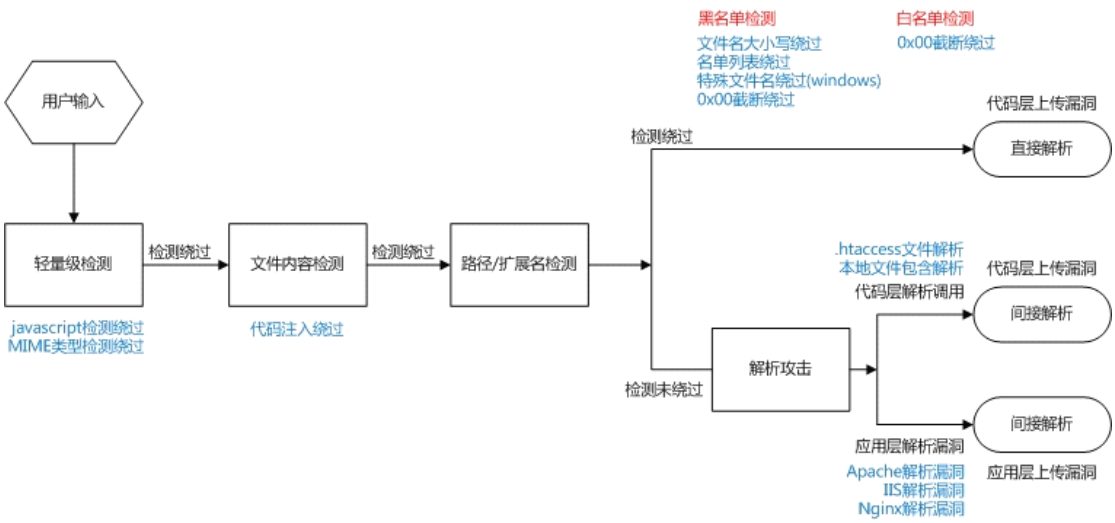


攻击要点:

要绕过轻量级检测
要绕过文件内容检测

A. 找到应用层的解析漏洞 (间接解析)

攻击手法与环节的对应如下图



[*] 这是整个上传攻击框架的核心之一

要注意下，在这里如果在路径/扩展名检测处检测未通过，流程到解析攻击时

白名单检测绕过技术里并不完全能利用所有解析攻击方式

在路径/扩展名检测绕过攻击里已经给出了具体细节

Fckeditor 2.4.x php 版的分析实例

On/Off	是否存在该检测	
Code/App	代码层还是应用层	
Safe/Vule	安全还是有漏洞	
Direct/Indirect	直接解析还是间接解析	
		Fckeditor 2.4.x 版本
轻量级检测		
[*] javascript检测		Off/Safe
禁用或直接提交		
[*] MIME类型检测		Off/Safe
伪造MIME类型		
文件内容检测		
[*] 文件加载检测		On/Vule
代码注入绕过		Code/Indirect
路径/扩展名检测		
[*] 黑名单检测		On/Vule
检测绕过		
文件名大小写绕过		
名单列表绕过		Code/Direct
特殊文件名绕过(windows)		Code/Direct
0x00截断绕过		Code/Direct
检测未绕过		
.htaccess文件解析		Code/Indirect
本地文件包含解析		Code/Indirect
Apache解析漏洞		App/Indirect
IIS解析漏洞		App/Indirect
Nginx解析漏洞		App/Indirect
[*] 白名单检测		On/Vule
检测绕过		
0x00截断绕过		Code/Direct
检测未绕过		
本地文件包含解析		Code/Indirect
IIS解析漏洞		App/Indirect
Nginx解析漏洞		App/Indirect

上传攻击分析框架

Code/App	代码层还是应用层
Safe/Vule	安全还是有漏洞
Direct/Indirect	直接解析还是间接解析
	待测试的代码与服务器环境
轻量级检测	
[*] javascript检测	
禁用或直接提交	
[*] MIME类型检测	
伪造MIME类型	
文件内容检测	
[*] 文件加载检测	
代码注入绕过	
路径/扩展名检测	
[*] 黑名单检测	
检测绕过	
文件名大小写绕过	
名单列表绕过	
特殊文件名绕过(windows)	
0x00截断绕过	
检测未绕过	
.htaccess文件解析	
本地文件包含解析	
Apache解析漏洞	
IIS解析漏洞	
Nginx解析漏洞	
[*] 白名单检测	
检测绕过	
0x00截断绕过	
检测未绕过	
本地文件包含解析	
IIS解析漏洞	
Nginx解析漏洞	

[*] 这是整个上传攻击框架的核心之二

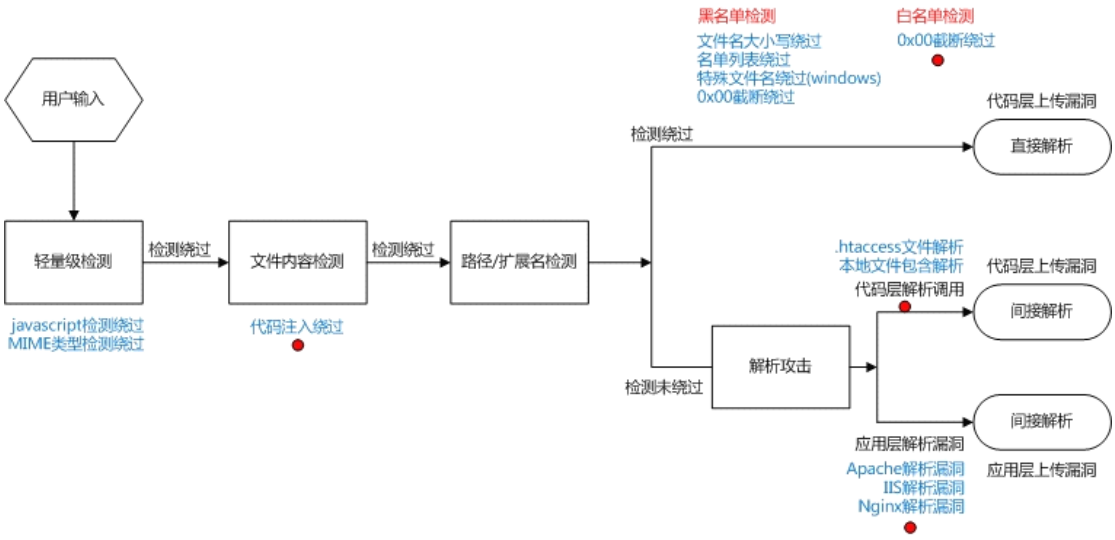
大家可以像上面分析 fckeditor 的方式，在分析一份源码或一个目标环境时
先把上面的上传攻击分析框架列表放好

然后去依次对比
哪些检测环节存在/不存在
哪些环节是安全/还是有漏洞
哪些环节如果被利用是代码层漏洞/还是 Web 应用程序解析漏洞
对应在该项后面进行填写
最后把 Vule 的部分选出来，再来分析如何进行组合，以及利用它们需要什么样的条件

通过这个分析框架进行白盒/黑盒分析并罗列出所有情况
攻击者便能更系统地分析出源码/目标环境可能存在的漏洞

整个上传攻击的核心最后被分析和总结成了上面两张重要的图
攻击手法与环节图 和 检测框架图
这两张图就是这篇 paper 最有价值的总结了

上传攻击的防御如下图



图中红点便是防御的重要环节
需要注意的:

1. 轻量级检测必然能绕过
2. 检测的重点放在文件内容检测
可以用检测脚本语言特征码的机制
3. 路径/扩展名检测一定要用白名单
并且注意路径的 0x00 截断攻击 (把 php 更新至最新版本即可，已经修补了这个漏洞了)
4. 不能有本地文件包含漏洞
5. 随时注意更新 web 应用软件
避免被解析漏洞攻击

- 结语

目前市面上所见过的文章或看过的书都没怎么看到过对上传攻击比较好的总结，见得最多的是对 sql injection 的总结，毕竟 sql injection 属于直接解析/执行的情况，效果来得最直接。但是这条路会越来越难走，总结其他攻击手段将会变得更加重要和有价值，像上传攻击总结，爆绝对路径的总结等等，一些平时大家当小技巧对待的东西，以后会成为主要攻击手段之一。

PS: 写这篇 paper 也写各种累死了，光是搭建各种环境就花了非常多时间，当然还是像文章开头自己所说的，希望分享能带来交流的价值，那么我的分享就是有意义的，毕竟大牛级别的一般不怎么分享他们比较高水平的技术，我也比较菜，没那么顾虑，就把这些小东西拿来分享下，给这个大环境注入点新鲜血液。