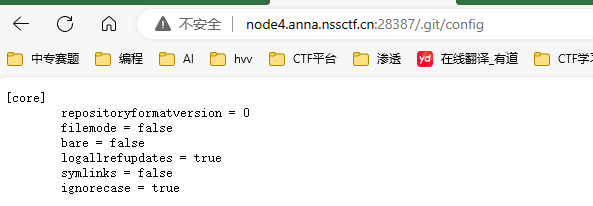
**敏感目录泄露**

**git泄露**

漏洞简介：git是一个主流的分布式控制系统，开发人员在开发过程中经常容易遗忘.git文件夹，导致攻击者通过.git文件夹中的信息获取开发人员提交过的所有源码，从而导致服务器受到致命攻击。

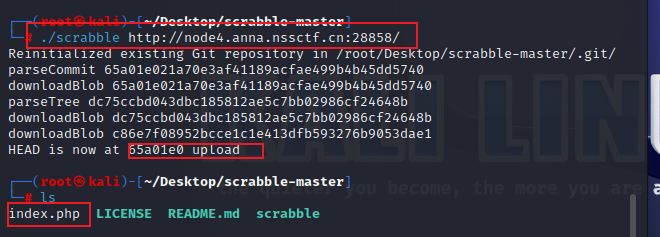
**常见git泄露**

初步测试就直接在后面加上/.git/config有回显说明有漏洞

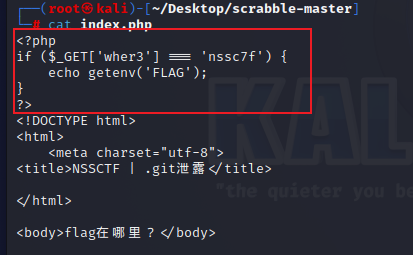


直接用工具和自己编写的脚步就可以获得flag，工具推荐<https://github.com/denny0223/scrabble>

使用方法就是直接./scrable 可疑网页



像这个照片就直接把index.php下载了下来而不是index.html。



**git回滚**

**git分支**

**git泄露的其他利用**

除了查看源码的常见的利用的方式，git泄露还有其他的利用的方式，如.git/config文件夹可能有access\_token信息，从而可以访问这给用户的其他仓库。

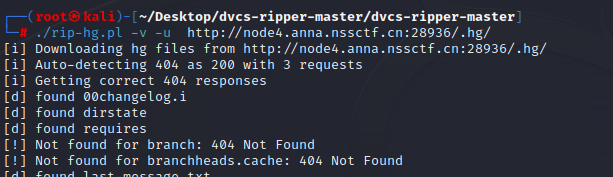
**SVN泄露**

SVN(subversion)源码版本管理软件，造成漏洞的原因是管理员操作不当，不规范将SVN隐藏文件夹暴露再外网，可以利用.svn/entries或wc.db文件获取到服务器的源码等信息，这里推荐两个工具，<https://github.com/kost/dvcs-ripper>，Seay-svn（windows源码备份利用工具）

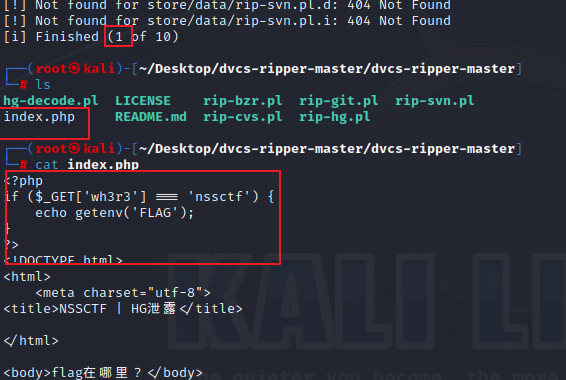
**HG泄露**

在初始化项目的HG会在当前文件下创建一个.hg的隐藏文件夹，其中包含源码和分支修改等信息，推荐工具<https://github.com/kost/dvcs>-ripper

使用方法例如：./rip-hg.pl -v -u <http://node4.anna.nssctf.cn:28936/.hg/> 这里要自己加/.hg/



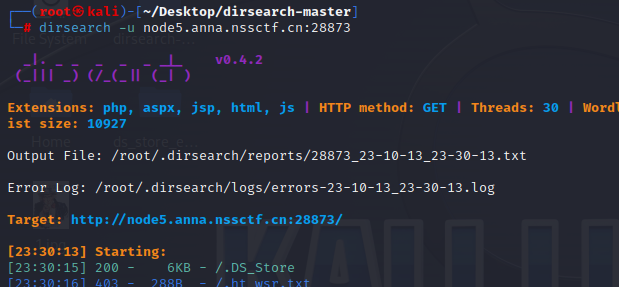
得到index.php



**DS\_Store泄漏**

.DS\_Store是Mac下Finder用来保存如何展示文件/文件夹的数据文件，每个文件夹队友下一个，由于开发和设计的失误，没删除文件中的隐藏的.DS\_Store，可能造成对目录结构的泄露源代码的泄露。

后台扫出

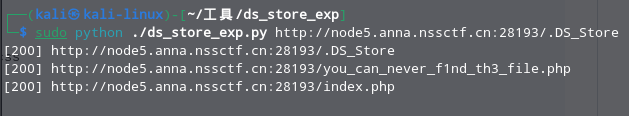


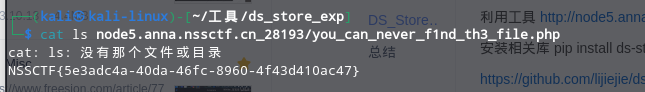
利用工具 <http://node5.anna.nssctf.cn:28873/>

安装相关库 pip install ds-store requests

<https://github.com/lijiejie/ds_store_exp>

python ds\_store\_exp.py <http://www.example.com/.DS_Store>





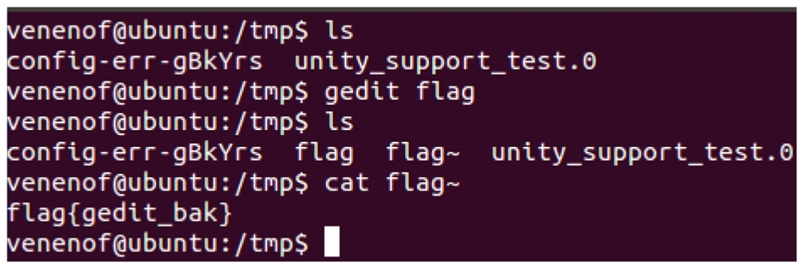
**总结**

不论是.git这些隐藏文件，还是其他的admin登录界面，关键在于字典的强大，推荐<https://github.com/maurosoria/dirsearch>，kali好像集成了可以直接下载dirsearch.

**敏感备份文件**

**gedit备份文件**

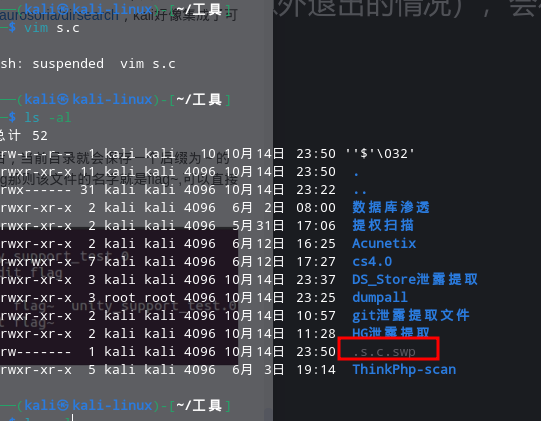
在linux 下使用gedit编辑器保存后，当前目录就会保存一个后缀为～的文件，比如刚编辑保存完成的flag那则该文件的名字就是flag~,可以直接从浏览器访问得到flag的源码



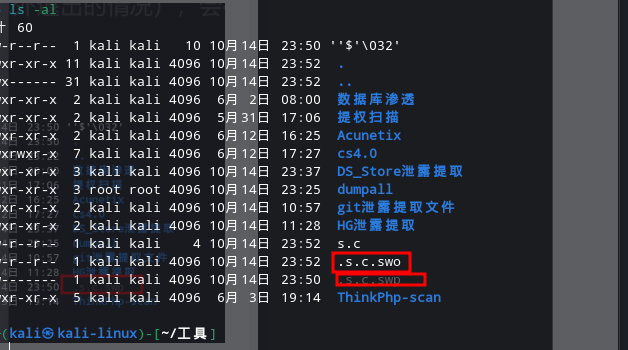
**Vim备份文件**

vim是目前linux中用的最多的编辑器，用户在意外退出或者通过服务连接vim写文件网络意外卡退都会产生一个备份文件。“.文件名字.swp”

我们直接用Ctrl+Z演示



如果意外退出了两次就会生成.文件名.swo



**常规文件**

robots.txt：记录一些目录和CMS版本信息

readme.md ：记录CMS版本信息，有的甚至是Github地址

www.zip/rar/tar.gz：往往是网站的备份

**总结经验**

在CTF线上比赛出题人往往会在线运维题目，有时导致SWP备份文件的生成，所以我们可以写脚本来监控题目的情况。

Vim第一次退出是.\*.swp,第二次是.\*.swo,第三次是.\*.swn

另外在实际环境中，网站备份文件往往可能是他的域名的压缩包。

**PHP<=7.4.21 Development Server源码泄露漏洞**

最近刷推特看到了一个洞，PHP<=7.4.21时通过

php -S开起的WEB[服务器](https://cloud.tencent.com/act/pro/promotion-cvm?from_column=20065&from=20065)存在源码泄露漏洞，可以将PHP文件作为静态文件直接输出源码，还蛮有意思的，这里大胆预测一波，最近在CTF里肯定会有人出这个点

**php中的非法传值**

当PHP版本小于8时，如果参数中出现中括号[，中括号会被转换成下划线\_，但是会出现转换错误导致接下来如果该参数名中还有非法字符并不会继转换成下划线\_，也就是说如果中括号[出现在前面，那么中括号[还是会被转换成下划线\_，但是因为出错导致接下来的非法字符并不会被转换成下划线\_

有一次CTF比赛就考了这个

**md5弱比较**

绕过这个

if($\_POST['a']!=$\_POST['b']&& md5($\_POST['a'])==md5($\_POST['b'])){

die("success!");

}

在弱比较里面，0e开头的会被识别为科学计数法，结果均为0，比较0=0为true绕过

**payload：**

a=QNKCDZO&b=s878926199a

常用[md5](https://so.csdn.net/so/search?q=md5&spm=1001.2101.3001.7020)加密后为0的字符串：

240610708，aabg7XSs，aabC9RqS

s878926199a

**md5强比较**

if($\_POST['a']!==$\_POST['b']&& md5($\_POST['a'])===md5($\_POST['b'])){

die("success!");

}

===强比较，如果传入的不是字符串而是数组，不但md5不会报错，结果是null在强比较null=null为ture绕过

a[]=1&b[]=2

**sha1弱比较**

弱比较都可以利用数组绕过

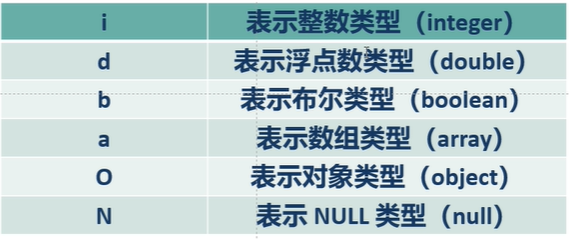
**序列化和反序列化漏洞**

序列化是指将数据结构例如数组，对象什么的，转换为一串字符流的过程可以使其在传输和缓存中持久化

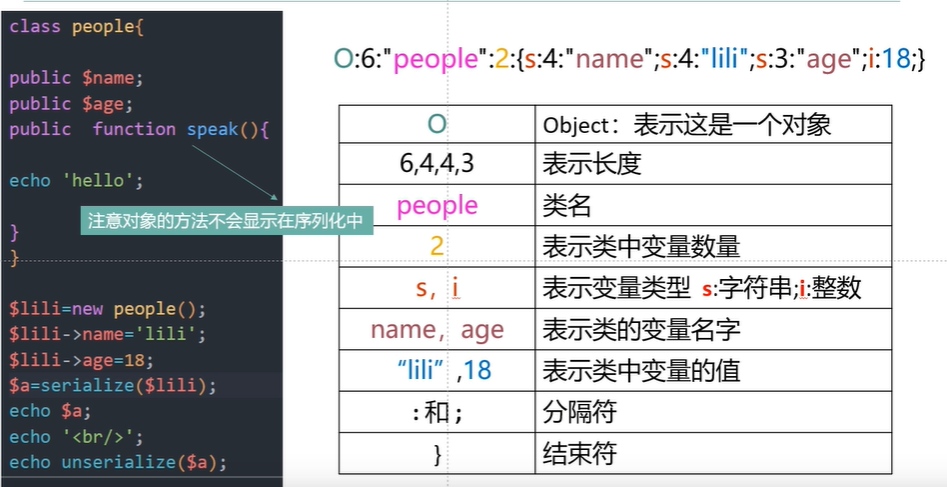
反序列化就是将一串序列化的字符流解码成对象或者数据结构，而这种还原的过程叫做反序列化

php中序列化的函数是 serialize() 反序列化就是 unserialize()

最前面的的代表，O是对象类型



**对象类型的序列化和反序列化**



原理实践

<?php

class people{

public $name;

public $age;

}

$shan=new people(); #对这个people这个类进行实体化等于shan

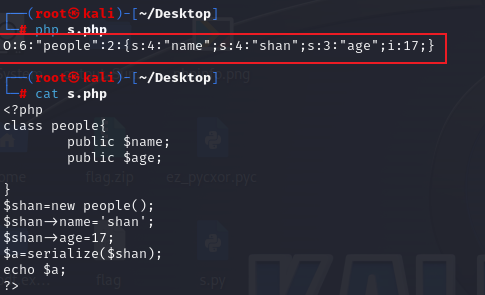
$shan->name='shan';

$shan->age=17;

$a=serialize($shan);

echo $a;

?>



O:6:"people":2:{s:4:"name";s:4:"shan";s:3:"age";i:17;}

O:类名长度:"类名":变量数目:{变量的类型:变量名的长度:"变量的名字";变量值的类型:变量值的长度:"变量的值";}

不同的属性序列化也不一样比如，private,protected

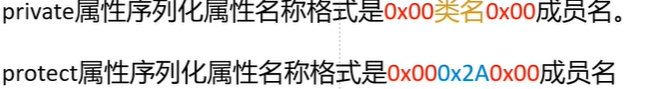
Pubic 公有

Private 私有

Protect 保护

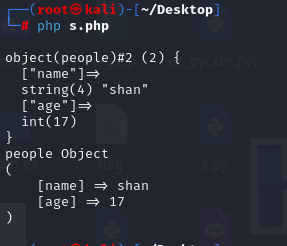
private $age=18; %00people%00age #people是类名

protected %height='156cm'; %00%2A%00height #%2A是\*



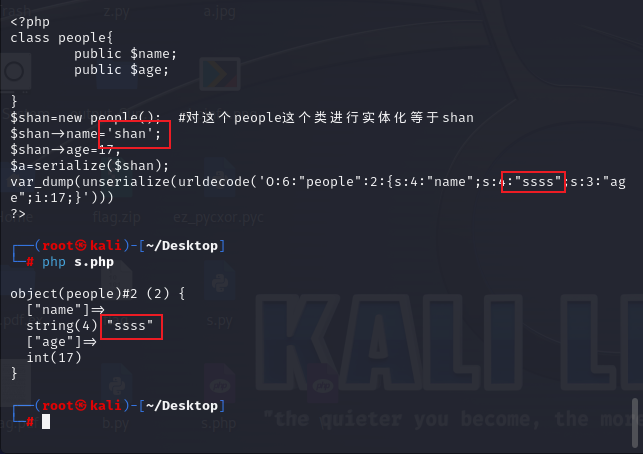
对象类型反序列化

如图上面这个是用var\_dump,下面使用print\_r

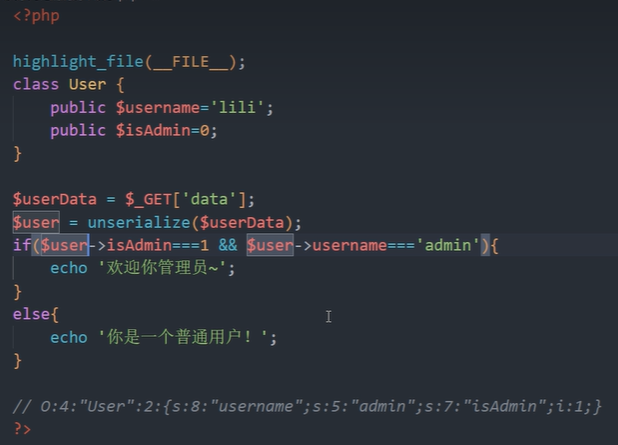


反序列化查看只能用var\_dump和print\_r

反序列化是不会看你之前的类而是看你要反序列化的数据的内容，就是说你可以在中途改掉之前序列化的内容而反序列化后的值会和你之前改掉的值一样而不是和原来还没序列化的值一样，如图：



初识反序列化漏洞



修改一下里面的值满足就直接是admin

**\_\_invoke()魔术方法 ： 将对象当作函数来使用时执行此方法**

**CVE-2016-7124 WAKEUP方法**

如果存在\_\_wakeup方法，调用 unserilize() 方法前则先调用\_\_wakeup方法，但是序列化字符串中表示对象属性个数的值大于 真实的属性个数时会跳过\_\_wakeup的执行

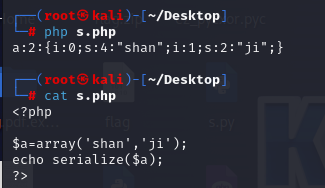
我们只需要把对象原来属性值比原来的大就行我们把1改成2

修改前:O:6:“sercet”:1:{s:12:“%00sercet%00file”;s:8:“flag.php”;}

修改后:O:6:“sercet”:2:{s:12:“%00sercet%00file”;s:8:“flag.php”;}



**数组类型序列化**

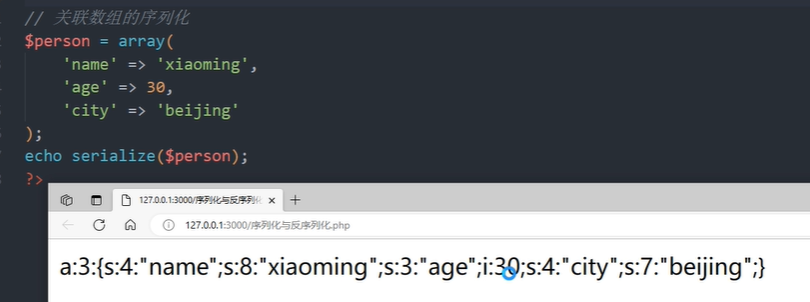


a:2:{i:0;s:4:"shan";i:1;s:2:"ji";}

a:数组中有效值的数量:{角标数字类型:0号角标;数值类型:数值长度:"数值内容";}

关联数组序列化

这个有点像Ｃ语言中的字典有键和值



**file\_get\_contents()函数**

源码：

if (file\_get\_contents($new\_player) === "Welcome to NSSCTF!!!")

因为file\_get\_content()函数是检查文件的，将文件内容都拿出来当值，所以我们不能直接传输值，而是要写到文件里去，这时候就要用data协议了

data://text/plain 是一种统一资源标识符（URI）方案，用于表示文本数据。它的原理是将文本数据直接嵌入到URI中，以便可以直接通过URI来引用和访问这些文本数据。

new\_player=data://text/plain,Welcome to NSSCTF ! ! !

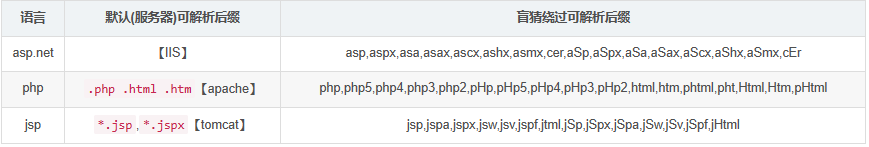
**任意文件上传漏洞**

PHP文件上传代码通常使用move\_upload\_file方法配合$file上传，如果是直接用上传者的后缀名而不进行判断和处理将存在漏洞

**文件黑名单绕过**

**使用特殊的后缀**

漏洞原理：没进行黑名单的漏网之鱼



**.htaccess解析**

漏洞原因：.htaccess文件是Apache服务器下的一个配置文件。其主要负责相关目录下的网页配置，即：在一个特定的文档目录中放置一个包含一个或多个指令的文件来对网页进行配置。

不过需要注意的是，.htaccess文件的作用域为其所在目录与其所有的子目录，不过若是子目录也存在.htaccess文件，则会覆盖父目录的.htaccess效果。

前提条件：[Apache](https://so.csdn.net/so/search?q=Apache&spm=1001.2101.3001.7020)开启.htaccess文件功能，可以上传.htaccess

最推荐的.htaccess内容，如果有限制就百度其他的内容

AddType application/x-httpd-php .jpg

这样一写就可以将我们上传的jpg当php解析

**大小写绕过**

漏洞原因：虽然设置了黑名单但是没有统一大小写，导致可以使用.pHp或者.PHp上传木马上去

**点绕过**

漏洞原因：windows系统下，文件后缀名最后一个点会被自动去除，而Linux系统下不会，所以只能在windows系统中生效，上传.php.

**空格绕过**

漏洞原因：文件上传系统不完善，没考虑到空格的存在，导致上传可以加入空格.php ，导致存入系统，如果是windows系统会将空格默认去位空，删除后面的空格，所以存储进去的文件就变成了.php后缀名导致木马上传，所以直接在后面加空格就可以上传木马

**::$DATA绕过**

漏洞原因：和上面的一样windows系统会自动去掉文件名后面的::$DATA从而绕过检查，存储的时候又自动删除::$DATA变成.php木马，所以直接在后面加::$DATA就可以上传木马

**双后缀名绕过**

漏洞原因：文件上传系统将上传的文件进行匹配匹配到的名单内后缀名就直接去除存储，上传a.php--->a然后就变成a了,所以我们直接上传.phphpp,去除一个我们直接重组成.php

**文件白名单绕过**

**前端JS绕过**

漏洞原因：只判断了类型是否为image/png等符合类型的文件

抓包修改类型为image/png等等

**%00截断绕过**

在C语言中 ”\0“是C语言中的结束符，如果用户能传入\0将可以实现截断

漏洞原理：00截断限制使用场景为：后端先获取用户上传的文件名如：1.php%00.jpg ，判断最后面是.jpg就成功放行储存，然后在储存的时候%00发生截断直接结束后面的字符导致存储成功的是1.php。

PHP的底层代码是C语言自然也存在这个问题，我们可以用到其他的语言场景中去。

这种情况常出现在ASP程序中，PHP 版本<5.3.4时也会有这个情况，JSP中也会出现。

**0x00绕过**

0x00是十六进制表示方法，表示ASCII码为0的字符，在一些函数处理时，会把这个字符当作结束符。

**0x0a绕过**

0xa是十六进制表示法，表示为ascii码则为\n换行符。

**突破getimagesize文件头绕过**

漏洞原因：检查文件内容的文件头的前几个字符，我们直接在文件内容前面加几个GIF89a就可以绕过

**二次渲染绕过**

漏洞原因：不相信上用户上传的任何文件，对上传上的文件进行重新生成，防止恶意代码。

绕过二次渲染方法：

将包含恶意代码的图片

phpinfo.gif

服务器上传图片功能代码

下载上传后的图片，并用

winhex.exe一句话木马

使用winhex对比功能，找到原图片与上传后图片的相同之处，并在该位置插入恶意代码；

实际操作：winhex——》工具——》比较——》搜索相同；展示区，蓝色部分是没有发生变化的,

在图片相同处，蓝色部分位置中间插入恶意代码，并上传，成功绕过；

然后使用文件包含来利用木马

**条件竞争绕过**

漏洞原理：先将上传的文件保存在服务器中，然后再进行对比白名单，如果不在白名单就删除，所以我们可以一边疯狂上传一边疯狂访问执行跳转木马

<?php fputs(fopen("info.php", "w"), '<?php @eval($\_POST["x"]);?>'); ?>

这个跳转木马就是访问内容为这个的木马就会生产名字为info.php的木马文件内容为后面的内容从而产生木马。

**CTF例题**

过滤了文件后缀名，mime类型，文件头，文件内容而且限制文件长度为15个字节及以下。

测试下来pht可以被正常解析，利用BM头和php中的反引号直接读取flag。

