

CS0424IT — ESERCITAZIONE S7L4

BUFFER OVERFLOW E SEGMENTATION FAULT

Simone La Porta



1 TRACCIA

Dato il seguente codice C che cerca di riprodurre un errore di segmentation fault a causa del buffer overflow, si cerca di riprodurre lo stesso errore aumentando la dimensione del parametro di input.

Il codice iniziale accetta in input una stringa di lunghezza massima di 10 caratteri:

```
#include <stdio.h>

int main() {
    char buffer[10];

    printf("Si prega di inserire il nome utente: ");
    scanf("%s", buffer);

    printf("Nome utente inserito: %s\n", buffer);

    return 0;
}
```

In sicurezza informatica, comprendere le vulnerabilità del software è cruciale per proteggere i sistemi dagli attacchi malevoli. Due delle vulnerabilità più comuni e pericolose sono il **buffer overflow** e il **segmentation fault**. Questi errori possono compromettere l'integrità e la sicurezza di un sistema.

2 BUFFER OVERFLOW

Il buffer overflow si verifica quando un programma tenta di scrivere più dati di quelli previsti in un'area di memoria riservata, chiamata buffer. Questo comportamento può sovrascrivere aree adiacenti di memoria, causando comportamenti imprevisti.

Cause

- Utilizzo di funzioni non sicure per la gestione di stringhe o dati, come `gets()`, `strcpy()`, `scanf("%s", ...)`.
- Mancanza di controllo sulla lunghezza dell'input fornito dall'utente.
- Assunzioni errate sulle dimensioni dei dati.

Conseguenze

- **Corruzione della memoria:** dati e codice possono essere sovrascritti, causando comportamenti imprevisti del programma.
- **Esecuzione di codice arbitrario:** gli attaccanti possono sfruttare un buffer overflow per iniettare ed eseguire codice maligno, ottenendo potenzialmente il controllo del sistema.
- **Crash del programma:** la sovrascrittura di dati critici può causare il crash del programma e portare ad esempio ad un Denial of Service (DoS).

3 SEGMENTATION FAULT

Un segmentation fault, o violazione di segmentazione, si verifica quando un programma tenta di accedere a una memoria che non gli è permesso di utilizzare. Questo comportamento è solitamente causato da un accesso a un puntatore non valido o da un buffer overflow.

Cause

- Accesso a puntatori non inizializzati o nulli.
- Dereferenziazione di puntatori invalidi o corrotti.
- Tentativi di accedere a memoria fuori dai limiti di un array o buffer.
- Buffer overflow che sovrascrivono l'indirizzo di ritorno di una funzione o altre strutture di controllo.

Conseguenze

- **Crash immediato del programma:** il sistema operativo rileva l'accesso non autorizzato e termina il programma per prevenire danni ulteriori.
- **Instabilità del sistema:** ripetuti segmentation fault possono causare instabilità nel sistema o nei servizi dipendenti dal programma affetto.
- **Debugging difficile:** individuare la causa esatta di un segmentation fault può essere complicato, richiedendo strumenti di debugging avanzati.

4 DIFFERENZE SOSTANZIALI

Natura del problema

- **Buffer Overflow:** è specificamente un problema di scrittura di dati oltre i limiti di un buffer.
- **Segmentation Fault:** è un problema di accesso a memoria non autorizzata, che può essere causato da vari fattori, incluso ma non limitato ai buffer overflow.

Conseguenze immediate

- **Buffer Overflow:** può corrompere la memoria e potenzialmente permettere l'esecuzione di codice arbitrario.
- **Segmentation Fault:** provoca tipicamente un crash immediato del programma, rilevato dal sistema operativo.

Scope di applicazione

- **Buffer Overflow:** è una vulnerabilità specifica che può essere sfruttata per attacchi di sicurezza.
- **Segmentation Fault:** è un sintomo di un problema più ampio di accesso alla memoria, che può o non può essere sfruttabile, ma indica sempre un bug nel programma.

Mentre un buffer overflow è una specifica vulnerabilità di sicurezza che implica la scrittura oltre i limiti di un buffer, un segmentation fault è un sintomo di accesso non autorizzato alla memoria, che può derivare da molteplici cause, incluso un buffer overflow. Entrambi sono critici in ambito di sicurezza informatica, ma il buffer overflow è direttamente sfruttabile per attacchi, mentre il segmentation fault è un indicatore di bug o vulnerabilità nel codice.

5 SVOLGIMENTO

In questo esercizio, si esamina come un buffer overflow può causare un segmentation fault in un programma C. Un buffer overflow si verifica quando un programma scrive più dati in un buffer di quanto ne possa contenere, sovrascrivendo aree adiacenti di memoria. Un segmentation fault si verifica quando il programma tenta di accedere a una regione di memoria non autorizzata.

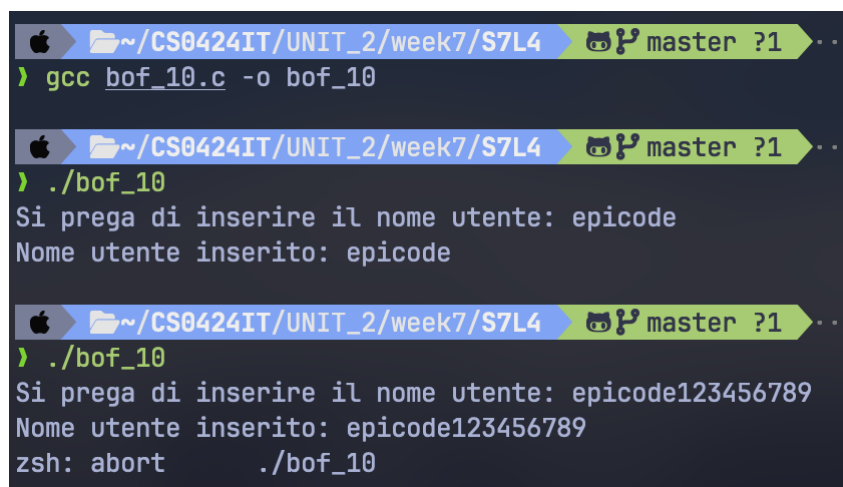
Codice iniziale

Il codice iniziale accetta un input di lunghezza massima di 10 caratteri.

```
1  #include <stdio.h>
2  int main() {
3      char buffer[10];
4
5      printf("Si prega di inserire il nome utente: ");
6      scanf("%s", buffer);
7
8      printf("Nome utente inserito: %s\n", buffer);
9
10     return 0;
11 }
```

Figura 1: Codice iniziale

L'esecuzione del codice iniziale mostra come un input di lunghezza superiore a 10 caratteri possa provocare un errore di segmentation fault a causa di buffer overflow.



```
Apple > ~/CS0424IT/UNIT_2/week7/S7L4 master ?1 ...
> gcc bof_10.c -o bof_10

Apple > ~/CS0424IT/UNIT_2/week7/S7L4 master ?1 ...
> ./bof_10
Si prega di inserire il nome utente: epicode
Nome utente inserito: epicode

Apple > ~/CS0424IT/UNIT_2/week7/S7L4 master ?1 ...
> ./bof_10
Si prega di inserire il nome utente: epicode123456789
Nome utente inserito: epicode123456789
zsh: abort      ./bof_10
```

Figura 2: Esecuzione del codice iniziale con input di lunghezza inferiore/superiore a 10 caratteri

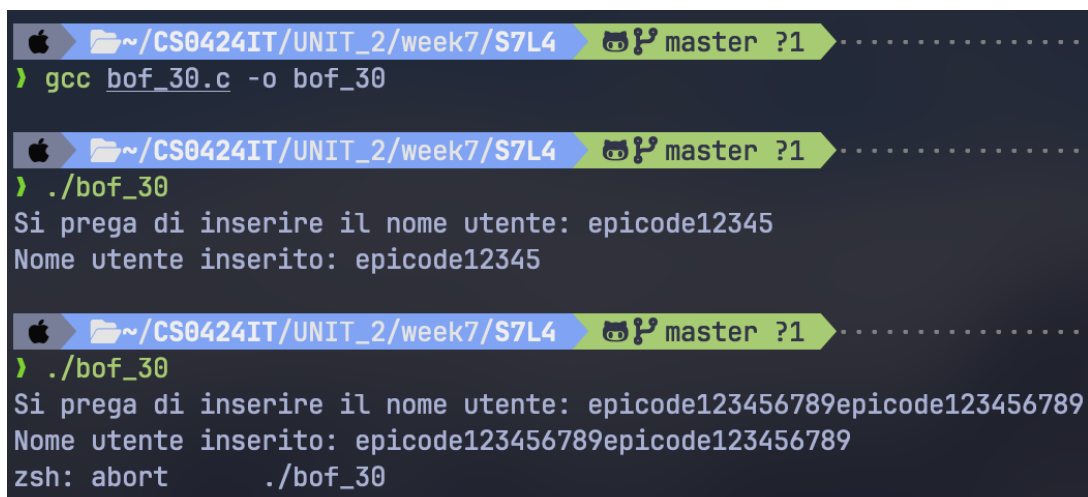
Codice modificato

Il codice viene modificato per accettare un input di lunghezza massima di 30 caratteri.

```
1  #include <stdio.h>
2  int main() {
3      char buffer[30];
4
5      printf("Si prega di inserire il nome utente: ");
6      scanf("%s", buffer);
7
8      printf("Nome utente inserito: %s\n", buffer);
9
10     return 0;
11 }
```

Figura 3: Codice modificato

Il codice viene compilato ed eseguito anche in questo caso, mostrando come un input di lunghezza superiore a 30 caratteri possa provocare un errore di segmentation fault a causa di buffer overflow.



```
Apple ~ /CS0424IT/UNIT_2/week7/S7L4 master ?1
> gcc bof_30.c -o bof_30

Apple ~ /CS0424IT/UNIT_2/week7/S7L4 master ?1
> ./bof_30
Si prega di inserire il nome utente: epicode12345
Nome utente inserito: epicode12345

Apple ~ /CS0424IT/UNIT_2/week7/S7L4 master ?1
> ./bof_30
Si prega di inserire il nome utente: epicode123456789epicode123456789
Nome utente inserito: epicode123456789epicode123456789
zsh: abort      ./bof_30
```

Figura 4: Esecuzione del codice modificato con input di lunghezza inferiore/superiore a 30 caratteri

6 CODICE SICURO

Di seguito è riportato un codice C che risolve le vulnerabilità di buffer overflow e segmentation fault, utilizzando funzioni sicure per la gestione degli input dell'utente e stampando un messaggio se l'input è troppo lungo:

```
1  #include <stdio.h>
2  #include <string.h>
3  #define BUFFER_SIZE 10
4
5  int main() {
6      char buffer[BUFFER_SIZE];
7      char extra; // Variabile per controllare se l'input è troppo lungo
8
9      printf("Si prega di inserire il nome utente: ");
10     // Utilizziamo fgets per limitare l'input alla dimensione del buffer
11     if (fgets(buffer, BUFFER_SIZE, stdin) != NULL) {
12         // Controlliamo se l'input è troppo lungo
13         if (buffer[strlen(buffer) - 1] != '\n') {
14             // L'input è troppo lungo se fgets non legge il carattere newline
15             printf("Errore: l'input è troppo lungo. Si prega di inserire un massimo "
16                 "di %d caratteri.\n",
17                 BUFFER_SIZE - 1);
18             // Puliamo il buffer di input per evitare problemi con gli input
19             // successivi
20             while ((extra = getchar()) != '\n' && extra != EOF) {
21                 // Scartiamo i caratteri extra
22             }
23         } else {
24             // Rimuoviamo il carattere di newline
25             buffer[strlen(buffer) - 1] = '\0';
26             printf("Nome utente inserito: %s\n", buffer);
27         }
28     } else {
29         printf("Errore nella lettura dell'input.\n");
30     }
31
32     return 0;
33 }
```

Figura 5: Esempio di codice sicuro

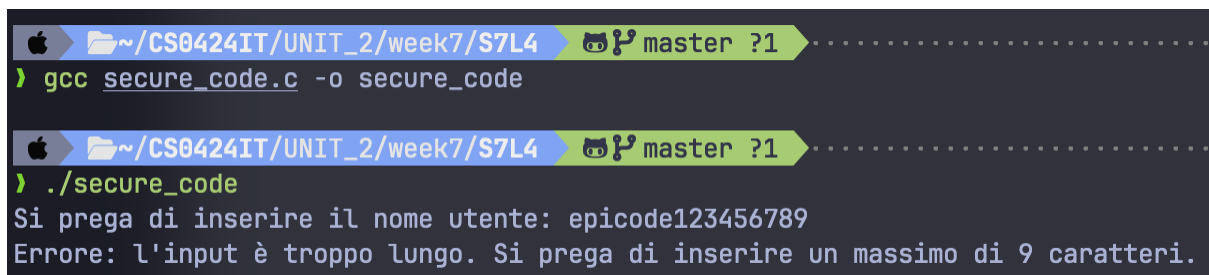
Il codice sicuro presentato adotta diverse misure per prevenire le vulnerabilità di buffer overflow e segmentation fault. Le principali caratteristiche di sicurezza includono:

- **Uso di fgets:** la funzione fgets è utilizzata per leggere l'input dell'utente in modo sicuro, limitando la quantità di dati letti alla dimensione del buffer. Questo previene il buffer

overflow controllando il numero massimo di caratteri che possono essere inseriti.

- **Verifica della lunghezza dell'input:** il codice controlla se l'input supera la dimensione del buffer verificando la presenza del carattere newline (`\n`). Se l'input è troppo lungo, viene stampato un messaggio di errore e i caratteri extra vengono scartati.
- **Rimozione del carattere newline:** se l'input è valido e contiene un carattere newline, questo viene rimosso per evitare che influenzi ulteriori elaborazioni. Questo garantisce che il buffer contenga solo i dati previsti dall'utente.
- **Pulizia del buffer di input:** in caso di input troppo lungo, il codice pulisce il buffer di input scartando i caratteri extra. Questo previene problemi con input successivi e garantisce la stabilità del programma.
- **Gestione degli errori di lettura:** il codice verifica se `fgets` ha letto correttamente l'input. In caso di errore, viene stampato un messaggio di errore appropriato. Questo assicura che il programma gestisca correttamente eventuali problemi di lettura dell'input.

Queste caratteristiche di sicurezza migliorano la robustezza del codice e prevengono le vulnerabilità comuni associate alla gestione degli input dell'utente.



```
~/CS0424IT/UNIT_2/week7/S7L4 master ?1  
» gcc secure_code.c -o secure_code  
  
~/CS0424IT/UNIT_2/week7/S7L4 master ?1  
» ./secure_code  
Si prega di inserire il nome utente: epicode123456789  
Errore: l'input è troppo lungo. Si prega di inserire un massimo di 9 caratteri.
```

Figura 6: Esecuzione del codice sicuro

L'esercizio ha dimostrato come un buffer overflow può causare un segmentation fault. Modificando la dimensione del buffer nel codice C, è possibile osservare il comportamento del programma con input di diverse lunghezze. Un input che eccede la dimensione del buffer provoca la sovrascrittura della memoria adiacente, portando a un segmentation fault. Questo evidenzia l'importanza di validare la lunghezza degli input e di utilizzare funzioni sicure per la gestione delle stringhe per prevenire vulnerabilità di sicurezza. Il codice sicuro presentato mostra come prevenire queste vulnerabilità, utilizzando tecniche di gestione sicura degli input.