INDICE

INTRODUZIONE ED OBIETTIVO

INTRODUZIONE: implementazione di un programma in linguaggio C che gestisce l'input dell'utente applicando controlli espliciti di validazione e sanitizzazione.
L'attività si concentra sull'analisi del comportamento del programma in presenza di input potenzialmente non conforme, mostrando come una gestione sicura dei dati in ingresso contribuisca a prevenire errori e comportamenti indesiderati, andando oltre la sola stabilità dell'applicazione.

OBIETTIVO: implementare controlli di validazione e sanitizzazione dell'input, garantendo che solo dati conformi vengano accettati dal programma, e dimostrare la differenza tra una semplice prevenzione del buffer overflow e un approccio più completo orientato al secure coding.

```
  GNU nano 8.7
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define BUF_SIZE 30

int main(void) {
    char buffer[BUF_SIZE];

    /* Lettura input in modo sicuro (previene BOF) */
    printf("Si prega di inserire il nome utente: ");
    if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
        printf("Errore di lettura input.\n");
        return 1;
    }

    /* Sanitizzazione: rimozione del newline finale */
    buffer[strcspn(buffer, "\n")] = '\0';

    /* Controllo: input vuoto */
    if (buffer[0] == '\0') {
        printf("Input non valido: stringa vuota.\n");
        return 1;
    }

    /* Sanitizzazione: accetta solo caratteri consentiti (lettere, numeri, underscore, trattino) */
    for (size_t i = 0; buffer[i] != '\0'; i++) {
        unsigned char c = (unsigned char)buffer[i];
        if (!(isalnum(c) || c == '_' || c == '-')) {
            printf("Input non valido: carattere non consentito.\n");
            return 1;
        }
    }

    printf("Nome utente inserito: %s\n", buffer);
    return 0;
}
```

In questa fase il programma è stato esteso introducendo controlli espliciti di sicurezza sulla gestione dell'input dell'utente.
La lettura dei dati avviene tramite una funzione sicura che limita automaticamente il numero di caratteri acquisiti in base alla dimensione del buffer, prevenendo qualsiasi possibilità di buffer overflow; successivamente, l'input viene sanitizzato rimuovendo il carattere di newline finale, al fine di normalizzare la stringa prima del suo utilizzo.
Sono stati inoltre implementati controlli di validazione per verificare che l'input non sia vuoto e che contenga esclusivamente caratteri consentiti.
In particolare, il programma accetta solo lettere, numeri, underscore e trattino, rifiutando qualsiasi altro carattere. In presenza di input non conforme, l'esecuzione viene interrotta e viene restituito un messaggio di errore appropriato.
Questo approccio consente di garantire non solo la sicurezza della memoria, ma anche la correttezza logica dei dati elaborati dal programma, applicando principi di secure coding orientati alla prevenzione e alla robustezza dell'applicazione.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./BOF

Si prega di inserire il nome utente: ajeje
Nome utente inserito: ajeje

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./BOF

Si prega di inserire il nome utente: fsdjvbswifjwiufhwuivbsufguhdhgfshudugudus
Nome utente inserito: fsdjvbswifjwiufhwuivbsufguhdh

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./BOF

Si prega di inserire il nome utente: mona17
Nome utente inserito: mona17

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./BOF

Si prega di inserire il nome utente: qwertyuiopasdfghjklzxcvbnmqwerasdfghjkl
Nome utente inserito: qwertyuiopasdfghjklzxcvbnmqwe
```

Rispetto alla versione precedente del programma, nella quale la prevenzione del buffer overflow era affidata esclusivamente alla limitazione della lunghezza dell'input, l'implementazione attuale introduce un livello di sicurezza aggiuntivo basato sulla validazione e sanitizzazione dei dati in ingresso: in entrambi i casi il programma non genera più errori di segmentazione, tuttavia il comportamento nei confronti dell'input risulta sostanzialmente diverso.
Nella versione precedente, infatti, qualsiasi stringa veniva accettata purché rientrasse nei limiti dimensionali del buffer, indipendentemente dal contenuto.
Nella versione attuale, invece, l'input viene sottoposto a controlli che ne verificano la conformità, consentendo l'elaborazione esclusivamente di stringhe valide e rifiutando quelle che contengono caratteri non ammessi o risultano vuote.
Questo approccio consente di distinguere tra una semplice protezione della memoria e una gestione sicura e consapevole dei dati, migliorando l'affidabilità complessiva del programma e riducendo ulteriormente il rischio di comportamenti indesiderati.

CONCLUSIONE

Questo laboratorio ha permesso di estendere il programma introducendo un approccio più completo alla gestione dell'input dell'utente, andando oltre la sola prevenzione del buffer overflow.

Attraverso l'implementazione di controlli di validazione e sanitizzazione, il programma è in grado di accettare esclusivamente dati conformi e sicuri, rifiutando input non valido prima che venga elaborato. I test eseguiti dimostrano che, anche in presenza di stringhe molto lunghe o di contenuti potenzialmente indesiderati, l'applicazione mantiene un comportamento stabile e prevedibile.

Questo risultato evidenzia come una corretta gestione dell'input rappresenti un elemento fondamentale del secure coding, contribuendo non solo alla sicurezza della memoria, ma anche all'affidabilità logica complessiva del software.