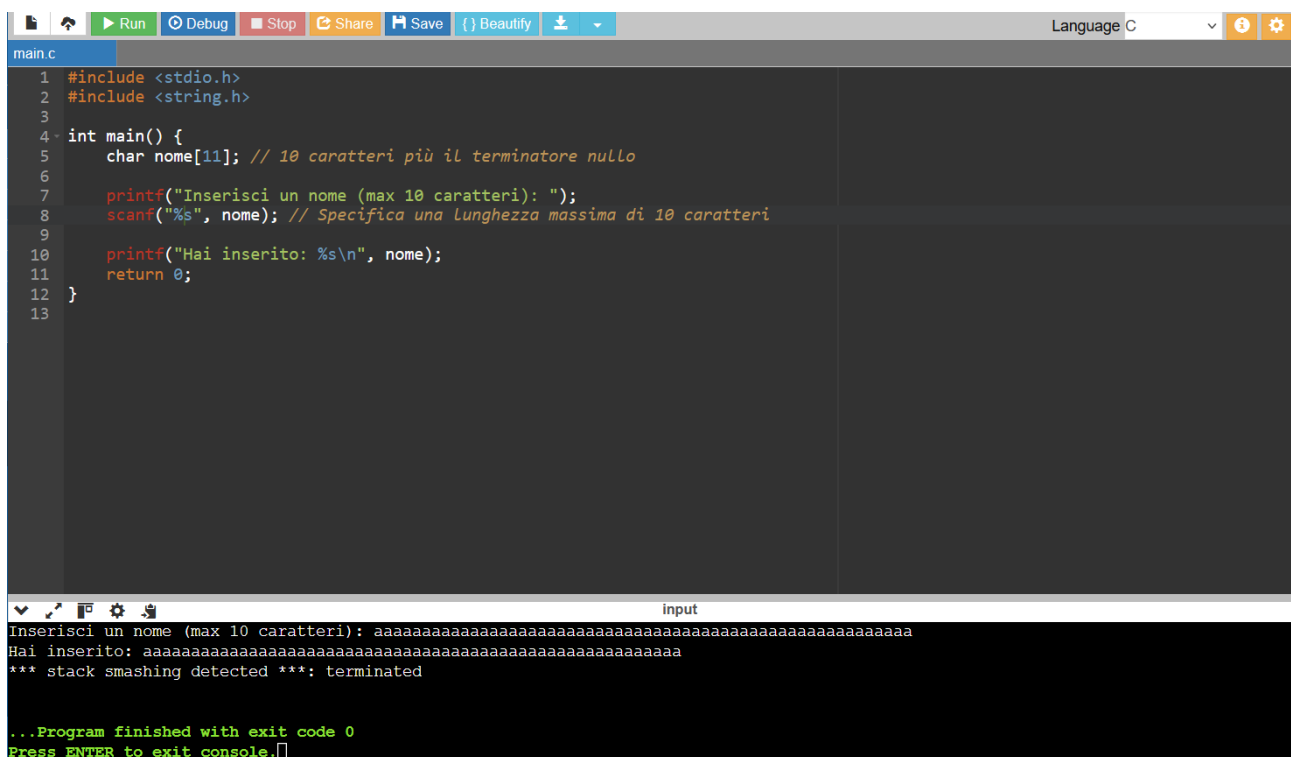


Traccia:

Nella lezione dedicata agli attacchi di sistema, abbiamo parlato dei buffer overflow, una vulnerabilità che è conseguenza di una mancanza di controllo dei limiti dei buffer che accettano input utente.

Nelle prossime slide vedremo un esempio di codice in C volutamente vulnerabile ai BOF, e come scatenare una situazione di errore particolare chiamata «segmentation fault», ovvero un errore di memoria che si presenta quando un programma cerca inavvertitamente di scrivere su una posizione di memoria dove non gli è permesso scrivere (come può essere ad esempio una posizione di memoria dedicata a funzioni del sistema operativo).

Un semplice programma in C dove chiede il nome utente:



```
main.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char nome[11]; // 10 caratteri più il terminatore nullo
6
7     printf("Inserisci un nome (max 10 caratteri): ");
8     scanf("%s", nome); // Specifica una lunghezza massima di 10 caratteri
9
10    printf("Hai inserito: %s\n", nome);
11    return 0;
12 }
13
```

```
input
Inserisci un nome (max 10 caratteri): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Hai inserito: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: terminated

...Program finished with exit code 0
Press ENTER to exit console.
```

Cosa succede quando si inseriscono 30 caratteri

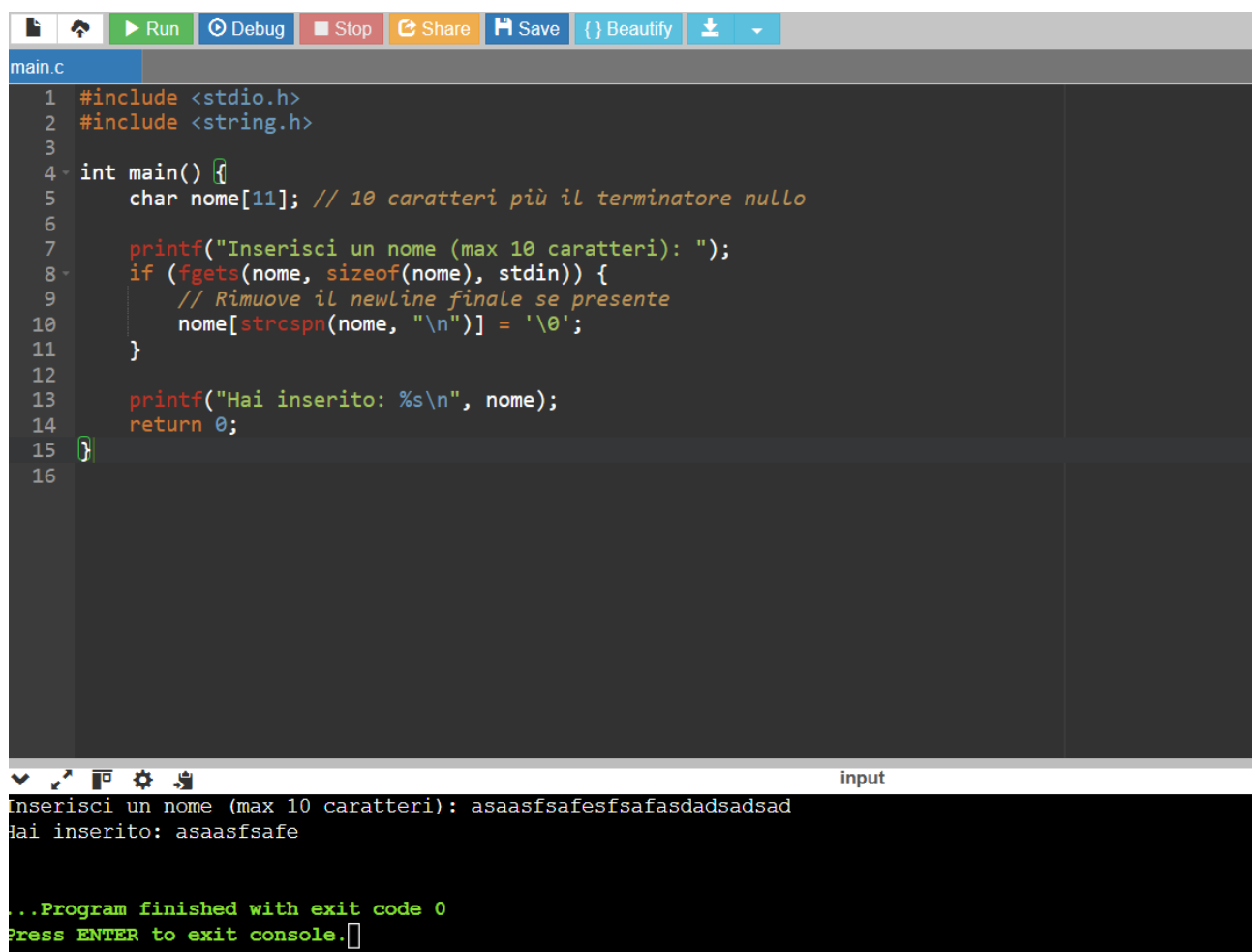
Quando si eseguono questo programma e si inseriscono 30 caratteri, possono verificarsi i seguenti problemi:

1. **Buffer Overflow:** Se scanf non ha il limite specificato (esempio %s come nel mio programma invece di %10s), inserendo più di 10 caratteri si supera la dimensione dell'array nome, causando un buffer overflow. Questo può sovrascrivere altre variabili in memoria, portando a comportamenti imprevedibili, crash del programma, o potenziali vulnerabilità di sicurezza.
2. **Sicurezza:** Anche se viene specificato %10s, l'input dell'utente che supera i 10 caratteri viene troncato. Tuttavia, l'utente potrebbe ancora tentare di sfruttare l'overflow del buffer se ci fossero altre vulnerabilità nel programma.

Come rimediare a questo buffer overflow

Per prevenire il buffer overflow, è essenziale limitare la quantità di dati letti e garantire che l'input non superi la dimensione dell'array riservato. Utilizzare funzioni di input sicure come `fgets` al posto di `scanf` è una pratica raccomandata. `fgets` permette di specificare esattamente quanti caratteri leggere, inclusi i caratteri di terminazione.

Una versione modificata del programma che utilizza `fgets` risulterebbe più sicura da BO:



```
main.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char nome[11]; // 10 caratteri più il terminatore nullo
6
7     printf("Inserisci un nome (max 10 caratteri): ");
8     if (fgets(nome, sizeof(nome), stdin)) {
9         // Rimuove il newline finale se presente
10        nome[strcspn(nome, "\n")] = '\0';
11    }
12
13    printf("Hai inserito: %s\n", nome);
14    return 0;
15 }
16
```

input

```
Inserisci un nome (max 10 caratteri): asaasfsafesfsafasdsadsad
Hai inserito: asaasfsafe

...Program finished with exit code 0
Press ENTER to exit console.
```

Programma più sicuro:

- **fgets**: La funzione `fgets` legge una riga di testo dall'input, specificando il numero massimo di caratteri da leggere, inclusi il carattere di nuova linea e il terminatore nullo.
- **strcspn**: La funzione `strcspn(nome, "\n")` trova la posizione del carattere di nuova linea `\n` e lo sostituisce con il terminatore nullo `\0` per rimuovere il newline che `fgets` include.

Questo approccio garantisce che l'input dell'utente non superi i 10 caratteri e protegge il programma dai buffer overflow.

