

System Exploit BOF

System Exploit BOF

1. Descrizione del codice

1.1 Dichiarazione delle variabili

```
int vector[10], i, j, k;
int swap_var;
```

- `vector[10]`: Un array che può contenere 10 numeri interi. È utilizzato per immagazzinare i valori inseriti dall'utente.
- `i, j, k`: Variabili utilizzate nei cicli `for` per iterare sui vari elementi dell'array.
- `swap_var`: Variabile temporanea usata per effettuare lo scambio tra gli elementi durante l'ordinamento.

1.2 Messaggio di input e acquisizione dei dati

```
printf("Inserire 10 interi:\n");
for (i = 0; i < 10; i++) {
    int c = i + 1;
    printf("[%d]: ", c);
    scanf("%d", &vector[i]);
}
```

- `printf("Inserire 10 interi:\n")`: Stampa il messaggio che chiede all'utente di inserire 10 numeri interi.
- Il ciclo `for` si ripete 10 volte (da `i = 0` a `i = 9`), chiedendo all'utente di inserire un numero intero per ogni iterazione:
 - `int c = i + 1`: Definisce una variabile che rappresenta l'indice umano (1-based) per la visualizzazione dell'input.
 - `printf("[%d]: ", c)`: Stampa un prompt che indica quale numero l'utente deve inserire (ad esempio, `[1]:` per il primo numero).
 - `scanf("%d", &vector[i])`: Acquisisce un numero intero e lo memorizza nella posizione corrispondente dell'array `vector`.

1.3 Stampa del vettore originale

```
printf("Il vettore inserito e':\n");
for (i = 0; i < 10; i++) {
    int t = i + 1;
```

```

    printf("[%d]: %d", t, vector[i]);
    printf("\n");
}

```

- `printf("Il vettore inserito e':\n")`: Stampa il messaggio per segnalare che il programma mostrerà il vettore inserito.
- Il ciclo `for` scorre tutti gli elementi dell'array `vector` e stampa ogni valore inserito:
 - `int t = i + 1`: Definisce una variabile che rappresenta l'indice umano (1-based) per la visualizzazione.
 - `printf("[%d]: %d", t, vector[i])`: Stampa l'indice e il valore corrispondente dell'array `vector`.
 - `printf("\n")`: Aggiunge una nuova riga per formattare l'output.

1.4 Ordinamento del vettore con Bubble Sort

```

for (j = 0; j < 10 - 1; j++) {
    for (k = 0; k < 10 - j - 1; k++) {
        if (vector[k] > vector[k + 1]) {
            swap_var = vector[k];
            vector[k] = vector[k + 1];
            vector[k + 1] = swap_var;
        }
    }
}

```

- Il codice utilizza l'algoritmo di ordinamento Bubble Sort per ordinare i numeri in ordine crescente.
- Il ciclo esterno `for (j = 0; j < 10 - 1; j++)` si occupa di gestire le iterazioni. Ogni passaggio "spinge" l'elemento più grande alla fine del vettore.
- Il ciclo interno `for (k = 0; k < 10 - j - 1; k++)` confronta ogni coppia di numeri adiacenti e li scambia se sono nell'ordine sbagliato:
 - `if (vector[k] > vector[k + 1])`: Verifica se l'elemento corrente è maggiore di quello successivo.
 - `swap_var = vector[k]`: Memorizza temporaneamente il valore di `vector[k]`.
 - `vector[k] = vector[k + 1]`: Sostituisce il valore di `vector[k]` con quello di `vector[k + 1]`.
 - `vector[k + 1] = swap_var`: Assegna il valore memorizzato temporaneamente a `vector[k + 1]`.

1.5 Stampa del vettore ordinato

```

printf("Il vettore ordinato e':\n");
for (j = 0; j < 10; j++) {

```

```
int g = j + 1;
printf("[%d]:", g);
printf("%d\n", vector[j]);
}
```

- `printf("Il vettore ordinato e':\n")`: Stampa un messaggio che indica che il programma mostrerà il vettore ordinato.
- Il ciclo `for` scorre nuovamente l'array `vector` per stampare i numeri ordinati:
 - `int g = j + 1`: Crea una variabile per l'indice umano (1-based).
 - `printf("[%d]: %d", g, vector[j])`: Stampa l'indice umano e il valore ordinato corrispondente.
 - `printf("\n")`: Aggiunge una nuova riga per una corretta formattazione.

1.6 Fine del programma

```
return 0;
```

- Il programma termina correttamente restituendo 0, che è una convenzione per indicare che l'esecuzione è stata completata senza errori.

2. Test del Programma

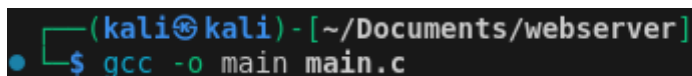
2.1 Obiettivo del Test

L'obiettivo del test è verificare il comportamento del programma in condizioni normali e confermare che funzioni correttamente per l'acquisizione e l'ordinamento dei numeri.

2.2 Esecuzione del Programma

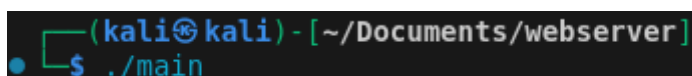
Il programma è stato compilato e eseguito su un ambiente Kali Linux. Il codice è stato salvato in un file chiamato `main.c` e compilato usando il compilatore `gcc` con il comando:

```
gcc -o main main.c
```



Poi, il programma è stato eseguito con il comando:

```
./main
```



2.3 Test di Funzionamento Normale

Quando l'utente inserisce esattamente 10 numeri, il programma:

- Stampa i numeri inseriti.
- Ordina correttamente i numeri in ordine crescente.
- Stampa il vettore ordinato.

```
(kali@kali) - [~/Documents/webserver]
$ ./main
Inserire 10 interi:
[1]:1
[2]:5
[3]:6
[4]:34
[5]:778
[6]:16
[7]:11
[8]:90
[9]:56
[10]:77
Il vettore inserito e':
[1]: 1
[2]: 5
[3]: 6
[4]: 34
[5]: 778
[6]: 16
[7]: 11
[8]: 90
[9]: 56
[10]: 77
Il vettore ordinato e':
[1]:1
[2]:5
[3]:6
[4]:11
[5]:16
[6]:34
[7]:56
[8]:77
[9]:90
[10]:778
```

3. Modifica del programma per indurre l'errore di segmentazione

La modifica del programma consisterà nell'aggiungere un ciclo che consente all'utente di inserire più di 10 numeri. Ad esempio, puoi cambiare la condizione del ciclo for che gestisce l'inserimento dei numeri in modo che consenta l'inserimento di più di 10 valori:

```

#include <stdio.h>

int main () {
    int vector[10], i, j, k;
    int swap_var;
    printf("Inserire più di 10 interi (ad esempio 15):\n");
    for (i = 0; i < 15; i++) {
        int c = i + 1;
        printf("[%d]: ", c);
        scanf("%d", &vector[i]);
    }
    printf("Il vettore inserito e':\n");
    for (i = 0; i < 15; i++) {
        int t = i + 1;
        printf("[%d]: %d\n", t, vector[i]);
    }
    printf("Provo a scrivere fuori dai limiti dell'array...\n");
    for (i = 10; i < 15; i++) {
        vector[i] = 9999;
    }
    int *ptr = (int *)0xDEADBEEF;
    printf("Provo a leggere da un indirizzo non valido...\n");
    printf("Valore letto dalla memoria non valida: %d\n", *ptr);
    for (j = 0; j < 15 - 1; j++) {
        for (k = 0; k < 15 - j - 1; k++) {
            if (vector[k] > vector[k + 1]) {
                swap_var = vector[k];
                vector[k] = vector[k + 1];
                vector[k + 1] = swap_var;
            }
        }
    }
    printf("Il vettore ordinato e':\n");
    for (j = 0; j < 15; j++) {
        int g = j + 1;
        printf("[%d]: %d\n", g, vector[j]);
    }
    return 0;
}

```

3.1 Differenze principali:

1. Inserimento di più di 10 valori:

- **Codice originale:** Si inseriscono solo 10 valori, come indicato dall'array `vector[10]`.
- **Codice modificato:** Si tenta di inserire più di 10 valori (15, ad esempio), cercando di scrivere fuori dai limiti dell'array.

2. Scrittura oltre i limiti dell'array:

- **Codice originale:** Non si verifica mai scrittura fuori dai limiti dell'array.
- **Codice modificato:** Scrittura esplicita oltre i limiti dell'array per cercare di forzare un errore.

3. Creazione di un puntatore non valido:

- **Codice originale:** Non viene mai utilizzato un puntatore non valido.

- **Codice modificato:** Si crea un puntatore che punta a una posizione di memoria arbitraria (`0xDEADBEEF`) e si tenta di dereferenziarlo per forzare un errore di segmentazione.

4. Stampa e ordinamento:

- **Codice originale:** Stampa e ordina correttamente i 10 valori.
- **Codice modificato:** Stampa e ordina anche i 15 valori, ma l'array potrebbe essere danneggiato da operazioni fuori limite, quindi il risultato potrebbe non essere quello atteso.
-

3.2 Risultato previsto:

Quando si esegue il programma, si dovrebbe ottenere un errore di segmentazione (segmentation fault) perché si sta cercando di accedere a indici dell'array che non sono stati allocati (oltre l'elemento `vector[9]`).

3.3 Risultato ottenuto:

```
(kali㉿kali) - [~/Documents/webserver]
$ ./main
Inserire più di 10 interi (ad esempio 15):
[1]:1
[2]:2
[3]:3
[4]:4
[5]:5
[6]:6
[7]:7
[8]:8
[9]:9
[10]:11
[11]:12
[12]:13
[13]:14
[14]:15
[15]:16
Il vettore inserito e':
[1]: 1
[2]: 2
[3]: 3
[4]: 4
[5]: 5
[6]: 6
[7]: 7
[8]: 8
[9]: 9
[10]: 11
[11]: 15
[12]: 12
[13]: 14
[14]: 15
[15]: 16
Provo a scrivere fuori dai limiti dell'array...
Provo a leggere da un indirizzo non valido...
zsh: segmentation fault ./main
```

Bonus

Come bonus per l'esercizio e' stato fatto:

1. **Controllo dell'input:** Per garantire che l'utente inserisca solo numeri validi e non vada oltre i limiti dell'array, possiamo aggiungere un ciclo di controllo dell'input che verifica che l'utente stia inserendo un intero valido.
2. **Menù per scegliere la modalità:** Aggiungeremo un menù che consente all'utente di decidere se eseguire la versione corretta del programma o quella che causerà l'errore di segmentazione.

```

#include <stdio.h>

int main() {
    int vector[10], i, j, k;
    int swap_var;
    int scelta;

    // Menù per scegliere la modalità
    printf("Scegli la modalità:\n");
    printf("1 - Esegui programma corretto\n");
    printf("2 - Forza errore di segmentazione\n");
    printf("Inserisci la tua scelta: ");
    scanf("%d", &scelta);

    if (scelta == 1) {
        printf("Inserisci 10 interi:\n");
        for (i = 0; i < 10; i++) {
            int c = i + 1;
            printf("[%d]: ", c);

            while (scanf("%d", &vector[i]) != 1) {
                printf("Input non valido. Inserisci un numero intero: ");
                while (getchar() != '\n');
            }
        }
        printf("Il vettore inserito e':\n");
        for (i = 0; i < 10; i++) {
            int t = i + 1;
            printf("[%d]: %d\n", t, vector[i]);
        }
        for (j = 0; j < 10 - 1; j++) {
            for (k = 0; k < 10 - j - 1; k++) {
                if (vector[k] > vector[k + 1]) {
                    swap_var = vector[k];
                    vector[k] = vector[k + 1];
                    vector[k + 1] = swap_var;
                }
            }
        }

        printf("Il vettore ordinato e':\n");
        for (j = 0; j < 10; j++) {
            int g = j + 1;
            printf("[%d]: %d\n", g, vector[j]);
        }
    } else if (scelta == 2) {
        int *ptr = (int *)0xDEADBEEF;
        printf("Provo a leggere la memoria fuori dai limiti dell'array...\n");
        printf("Valore letto dalla memoria non valida: %d\n", *ptr);
    } else {
        printf("Scelta non valida.\n");
    }
    return 0;
}

```

Spiegazione delle modifiche:

1. Menù di scelta:

- Abbiamo introdotto una variabile `scelta` che consente all'utente di scegliere tra due modalità: la modalità corretta (1) o la modalità che causa l'errore (2).
- La selezione viene effettuata tramite un `scanf` che memorizza la scelta dell'utente.

2. Controllo dell'input:

- Abbiamo utilizzato un ciclo `while` per verificare che l'utente inserisca solo interi validi. Se l'utente inserisce qualcosa che non è un numero intero (come una lettera o un simbolo), il programma chiederà di inserire nuovamente un numero, ignorando ciò che è stato inserito erroneamente.
- La funzione `getchar()` è usata per pulire il buffer in caso di input non valido.

3. Modalità con errore di segmentazione:

- Quando l'utente sceglie la modalità che causa l'errore di segmentazione (`scelta == 2`), si tenta di accedere a una memoria non valida (`0xDEADBEEF`), il che causerà un errore di segmentazione.

Come funziona:

- Quando si esegue il programma, si vedrà un menù che chiederà se si vuole eseguire il programma normalmente o causare l'errore di segmentazione.
- Se si sceglie la modalità corretta, il programma chiederà di inserire 10 numeri interi, ordinerà l'array e lo visualizzerà.
- Se si sceglie la modalità con errore, il programma tenterà di accedere a una zona di memoria non valida, causando un errore di segmentazione.