Traccia Giorno 3

System Exploit BOF

Traccia Giorno 3:

https://drive.google.com/file/d/1nEM_FV5zFHj4hw9_Ya1PUP_xf5bLGy0I/view
Leggete attentamente il programma in allegato. Viene richiesto di:

- Descrivere il funzionamento del programma prima dell'esecuzione.
- Riprodurre ed eseguire il programma nel laboratorio - le vostre ipotesi sul funzionamento erano corrette?
- Modificare il programma affinché si verifichi un errore di segmentazione.

Suggerimento:

Ricordate che un BOF sfrutta una vulnerabilità nel codice relativo alla mancanza di controllo dell'input utente rispetto alla capienza del vettore di destinazione. Concentratevi quindi per trovare la soluzione nel punto dove l'utente può inserire valori in input, e modificate il programma in modo tale che l'utente riesca ad inserire più valori di quelli previsti.

```
#include <stdio.h>

int main () {

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


printf ("Inserire 10 interi:\n");

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


printf ("Il vettore inserito e':\n");
for ( i = 0 ; i < 10 ; i++)
        {
        int t= i+1;
        printf("[%d]: %d", t, vector[i]);
        printf("\n");
        }


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]:", g);
        printf("%d\n", vector[j]);
        }

return 0;


}
```

Spiegazione codice

Il programma funziona in questo modo:

1. **Inizializzazione delle variabili**
    - Viene dichiarato un array vector[10] di interi, che conterrà 10 elementi.
    - Sono dichiarate inoltre alcune variabili di supporto (i, j, k) e una variabile temporanea swap_var utilizzata per lo scambio di valori durante l'ordinamento.
2. **Acquisizione dei dati**

- Tramite un ciclo for, all'utente vengono richiesti in input 10 numeri interi, che vengono memorizzati negli indici 0–9 dell'array.

3. **Visualizzazione dell'array inserito**
    - Un secondo ciclo for stampa sullo schermo i valori dell'array nell'ordine in cui sono stati immessi dall'utente.
4. **Ordinamento (Bubble Sort)**
    - L'algoritmo di ordinamento implementato è il *bubble sort*.
    - Si basa su **due cicli annidati**:
        - Il ciclo esterno (j) stabilisce il numero di passate sull'array.
        - Il ciclo interno (k) confronta coppie di elementi adiacenti vector[k] e vector[k+1].
    - Se il valore a sinistra è maggiore di quello a destra, i due vengono scambiati tramite swap_var.
    - Dopo ogni passata, l'elemento più grande tra quelli rimasti si "sposta" verso la parte destra dell'array (in posizione corretta).
    - Al termine delle iterazioni, l'array risulta ordinato in ordine crescente.
5. **Visualizzazione dell'array ordinato**
    - Infine, l'array viene ristampato, mostrando i valori riordinati.

Screenshot funzionamento (test programma)

Versione modificata che provoca un errore di segmentazione

Per generare un segmentation fault ho deliberatamente forzato un buffer overflow: ho dichiarato un array di 10 interi (vector[10]) e poi ho inserito un ciclo che itera fino a un numero estremamente grande (10^18), scrivendo valori oltre la capacità effettiva del buffer. In questo modo, già dall'undicesima iterazione, il programma scrive in memoria non allocata, corrompendo lo stack e causando un comportamento indefinito che porta all'errore di segmentation fault.

```
#include <stdio.h>


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


    printf("Inserire 10 interi:\n");
    for (i = 0; i < 10000000000000000000; i++) {
        int c = i + 1;
        printf("[%d]:", c);
        scanf("%d", vector[i]=c);
    }


    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++) {  // <= provoca accesso a vector[10]
            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]);
    }

    return 0;
}
```

Screenshot funzionamento programma con generazione errore segmentation fault.

Esercizio Bonus

Bonus

Inserire controlli di input

Creare un menù per far decidere all'utente se avere il programma che va in errore oppure quello corretto

Per farlo ho creato un metodo leggi_intero_sicuro e ho applicato il metodo ad un ciclo while infinito all'interno del for di riempimento dell'array che consente l'uscita (break) solo quando la condizione if è soddisfatta, questa condizione è che il numero inserito dall'utente rispetti le varie condizioni dentro il metodo leggi_intero _sicuro.

Le sue condizioni sono:

Descrizione della funzione leggi_intero_sicuro

La funzione leggi_intero_sicuro ha lo scopo di leggere da input un valore numerico intero, garantendo che l'input fornito dall'utente sia effettivamente valido e rientri nei limiti rappresentabili dal tipo int. Essa implementa una serie di controlli per prevenire errori comuni dovuti a input non corretti (ad esempio stringhe non numeriche, valori troppo grandi o troppo piccoli, caratteri residui dopo il numero).

Il funzionamento avviene nelle seguenti fasi:

1. **Lettura della stringa di input**:
   La funzione utilizza fgets per leggere fino a 127 caratteri dalla tastiera, memorizzandoli in un buffer (buf). In questo modo si evitano buffer overflow e si mantiene il controllo sulla dimensione massima dell'input.
2. **Rimozione del newline**:
   Eventuale carattere di fine riga (\n), inserito quando l'utente preme INVIO, viene sostituito con il terminatore '\0' per garantire che la stringa sia correttamente gestita dalle funzioni di conversione.
3. **Conversione in numero intero**:
   La stringa viene convertita in un valore di tipo long tramite la funzione strtol. Questa funzione fornisce un puntatore (end) che indica il primo carattere non utilizzato durante la conversione.
4. **Controlli sulla conversione**:
   - Se non sono state trovate cifre (end == buf) l'input viene considerato non valido.
   - Se si verifica overflow o underflow numerico (errno == ERANGE), l'input viene scartato.
   - Eventuali spazi o tabulazioni finali vengono ignorati, ma se restano caratteri residui (*end != '\0') l'input viene considerato errato.
5. **Verifica dei limiti del tipo int**:
   Anche se la conversione a long è riuscita, il valore viene accettato solo se rientra nei limiti previsti da INT_MIN e INT_MAX (definiti in <limits.h>).
6. **Restituzione del risultato**:
   Se tutti i controlli hanno avuto esito positivo, il valore convertito viene salvato nella

variabile passata per riferimento tramite *out. La funzione ritorna 1 per indicare successo, altrimenti ritorna 0 per segnalare un errore di input.

```c
static int leggi_intero_sicuro(int *out) {
    char buf[128];
    //fgets legge al massimo 127 caratteri + \0 (termine inserimento) (legge anche il numero che viene passato, una sorta di scanf)
    if (!fgets(buf, sizeof buf, stdin)) {
        return 0; // EOF/errore EOF significa fine del file (end of file)
    }

    // rimuovi newline se presente (strcspn(buf, "\n") trova l'idice del primo \n se c'è e lo cambia con \0 (\0 = al termine dell'inserimento di un dato))
    buf[strcspn(buf, "\n")] = '\0';

    // strtol con controlli errno viene usato dalle librerie C per segnalare il tipo d'errore verificato.
    errno = 0;
    // end è un puntatore di una cella di memoria in questo caso usato da strtol sul primo carattere non valido una volta letto il numero.
    char *end = NULL;
    // strtol converte il prefisso "numero" in base 10 ignora gli spazzi iniziali accetta il segno + o - se non ci sono cifre valide lascia end == buf, su overflow e underflow imposta errno = enrang
    // buf inizio stringa da convertire, end verra fatto puntare da strtol al primo carattere non valido trovato.
    // se end == buf allora la funzione non ha trovato nenache una cifra valida all'inizio della stringa esempio: buf = 'abc' neanche una cifra valida quindi buf == end
    // errno=enrange perchè strtol converte la stringa in un long. ma se il numero è troppo grande da problemi e ritornerà LONG_MAX se overflow positivo e LONG_MIN se negativo.
    long val = strtol(buf, &end, 10);
    // qui dice che se c'è uno di questi 2 ritorni allora la cifra non è valida per i motivi sopra elencati.
    if (errno == ERANGE || end == buf) {
        // overflow oppure nessuna cifra valida
        return 0;
    }
    // verifica che non restino caratteri non numerici
    // utilizando end (puntatore) finchè end punta spazzi questa parte di codice gli dice di andare avanti.
    while (*end == ' ' || *end == '\t') end++;
    // se subito dopo lo spazio però end non sta puntando \0 che rappresenta la fine della stringa allora returna 0 errore se la sta puntando la stringa va bene.
    if (*end != '\0') {
        return 0;
    }
    // controlla che rientri nel range int INT_MIN E INT_MAX sono dei cap per i numeri interi per non far mettere numeri infiniti o comunque che non superino i 32 bit.
    if (val < INT_MIN || val > INT_MAX) {
        return 0;
    }
// scriviamo il valore convertito nella variabile chiamata out e returniamo 1 quindi il numero ha superato i controlli.
    *out = (int)val;
    return 1;
}
```

```c
  printf("Inserire 10 interi:\n");
  for (i = 0; i < 10; i++) {
      //è un ciclo infinito continuerà a girare finche le condizioni del metodo leggi intero sicuro non vengono soddisfatte.
      while (1) {
          int c = i + 1;
          //variabile temporanea in cui viene memorizzato il valore letto.
          int tmp;
          printf("[%d]: ", c);
          // condizione secondo cui se le condizioni del metodo sono soddisfatte torna 1 e fa il break per inserire il prossimo numero o finire il ciclo
          // e 0 se non sono valide e quindi ricicla facendo inserire il numero nuovo.
          if (leggi_intero_sicuro(&tmp)) {
              vector[i] = tmp;
              break;
          } else {
              // stampa messaggio d'errore su stderr il ciclo non fa break quindi ricomincia da capo.
              fprintf(stderr, "Input non valido. Inserire un intero valido.\n");
          }
      }
  }
```

```
Inserire 10 interi:
[1]: ciao
Input non valido. Inserire un intero valido.
[1]: ciao123
Input non valido. Inserire un intero valido.
[1]: 1 2 3
Input non valido. Inserire un intero valido.
[1]: 99999999999999999999999999999999999999999999999999999999999999999999999999999999999999
Input non valido. Inserire un intero valido.
[1]: 10
[2]: 1
[3]: 2
[4]: 3
[5]: 4
[6]: 5
[7]: 6
[8]: 7
[9]: 8
[10]: 9
Il vettore inserito e':
[1]: 10
[2]: 1
[3]: 2
[4]: 3
[5]: 4
[6]: 5
[7]: 6
[8]: 7
[9]: 8
[10]: 9
Il vettore ordinato e':
[1]: 1
[2]: 2
[3]: 3
[4]: 4
[5]: 5
[6]: 6
[7]: 7
[8]: 8
[9]: 9
[10]: 10
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-pqesympc.3ly" 1>"/tmp/Microsoft-MIEngine-Out-amobjf21.jlr"
```

Dopo aver verificato il funzionamento di tutti e due aver triggerato appositamente l'errore, ho inserito tutto in un file e ho creato un menu dentro il main che richiamasse le altre 2 funzioni programma_sicuro e programma_vulnerabile.

Questo è il main:

```
int main(void) {
    int scelta;

    puts("=== MENU ===");
    puts("1) Programma Sicuro (controlli sugli input)");
    puts("2) Programma Vulnerabile (nessun controllo)");
    printf("Scelta: ");

    // Lettura scelta con input sicuro (riuso della funzione)
    if (!leggi_intero_sicuro(&scelta)) {
        fprintf(stderr, "Input non valido.\n");
        return 1;
    }

    if (scelta == 1) {
        programma_sicuro();
    } else if (scelta == 2) {
        programma_vulnerabile();
    } else {
        puts("Opzione non valida.");
        return 1;
    }

    return 0;
}
```

A schermo quello che si vede è questo(programma non sicuro):

```
===Menu===
1) Esegui programma Sicuro (controlli sugli input)
2) Esegui programma VUlnerabile (non ci sono controlli sugli input)
Scelta:2
Inserire 10 interi:
[1]: 2 3
[2]: [3]: 4 5 6
[4]: [5]: [6]: ciao
[7]: [8]: [9]: [10]: Il vettore inserito e':
[1]: 2
[2]: 3
[3]: 4
[4]: 5
[5]: 6
[6]: 0
[7]: 0
[8]: 0
[9]: -9208
[10]: 32767
Il vettore ordinato e':
[1]: -9208
[2]: 0
[3]: 0
[4]: 0
[5]: 2
[6]: 3
[7]: 4
[8]: 5
[9]: 6
[10]: 32767
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm}
.2nr"
```

A schermo è questo(programma sicuro):

```
1) Esegui programma Sicuro (controlli sugli input)
2) Esegui programma VUlnerabile (non ci sono controlli sugli input)
Scelta:1
Inserire 10 interi:
[1]: ciao
Input non valido. Inserire un intero valido.
[1]: 123 carlo
Input non valido. Inserire un intero valido.
[1]: 1 2 3
Input non valido. Inserire un intero valido.
[1]:
Input non valido. Inserire un intero valido.
[1]: 1
[2]: 34
[3]: 55
[4]: 44
[5]: 33
[6]: 22
[7]: 11
[8]: 10
[9]: 66
[10]: 88
Il vettore inserito e':
[1]: 1
[2]: 34
[3]: 55
[4]: 44
[5]: 33
[6]: 22
[7]: 11
[8]: 10
[9]: 66
[10]: 88
Il vettore ordinato e':
[1]: 1
[2]: 10
[3]: 11
[4]: 22
[5]: 33
[6]: 34
[7]: 44
[8]: 55
[9]: 66
[10]: 88
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-ls5g5gls.xdh" 1>"/tmp/Microsoft-MIEngine-O
.j2n"
```