Build Week 2: esercizio giorno 3

La consegna dell'esercizio di oggi prevedeva l'analisi di un codice in C, comprenderne la funzione ed eseguirlo, ed in seguito modificarlo in modo tale che si venisse a creare un errore di segmentazione dovuto ad un buffer overflow.
Come bonus: inserire dei controlli su l'input dell'utente e modificare il codice in modo tale da dare una scelta all'utente se utilizzare lo script corretto o quello vulnerabile.

Di seguito uno screen che mostra il codice fornitoci.

```
#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;


}
```

Senza eseguire il codice, abbiamo già capito più o meno la sua funzione: già solo dalle scritte che il programma stampa a schermo si intuisce che l'utente dovrà inserire 10 interi, che verranno inseriti in un vettore che verrà poi riordinato tramite un algoritmo.
Andando a guardare più a fondo si vede come in effetti venga inizialmente definito come variabile un array che può contenere 10 interi. Inoltre i numeri che l'utente inserirà verranno stampati a schermo a fianco del loro progressivo di inserimento da 1 a 10 (mentre gli indici dell'array vanno da 0 a 9), e il programma stamperà a schermo il nostro vettore "disordinato" prima di riordinarlo.

```c
#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");
        }
```

Poi c'è il cuore del codice, ovvero l'algoritmo di riordinamento.
L'algoritmo si basa su due cicli, uno esterno ed uno interno.
Quello interno confronta a due a due gli elementi del vettore da sinistra verso destra e sposta a ogni sua esecuzione il numero più grande trovato il più a destra possibile: infatti se alla prima esecuzione il numero più grande dell'array sarà spostato nell'ultima posizione, alla seconda il secondo numero più grande dell'array verrà spostato nella penultima posizione, e così via.
Quello esterno controlla il numero di esecuzioni del ciclo interno.
Capiamo quindi che il programma ordina in modo crescente gli elementi del nostro vettore di input.
Tale algoritmo viene chiamato "bubble sort", ovvero "ordinamento di una bolla".

```c
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;
                        }
                }
        }
```

Infine il programma stamperà a schermo il vettore ordinato.

Eseguendo il codice abbiamo avuto una conferma di quella che è stata la nostra interpretazione.

Ora dobbiamo vedere come modificare il codice per causare un errore di segmentazione.

Una prima idea, potrebbe essere molto banalmente quella di modificare l'argomento della funzione `scanf` da `scanf("%d",&vector[i])` in `scanf("%d",vector[i])`, ovvero togliere il carattere "&".
Infatti se `vector[i]` è l'i-esimo elemento del vettore, `&vector[i]` è il suo indirizzo di memoria, ed è questo che bisogna inserire nell'argomento della funzione `scanf`.
Questo sostanzialmente farebbe leggere alla funzione il nostro intero come se fosse un indirizzo di memoria, andando quindi a tentare di scrivere dati in una locazione di memoria che molto probabilmente non appartiene al nostro programma e a cui quindi lo stesso non ha permesso di accedere, causando un comportamento anomalo del programma, che potrebbe essere anche un ciclo infinito, ma con buona probabilità farebbe intervenire il sistema operativo con la generazione di un errore di segmentazione.

La traccia però suggeriva di usare una vulnerabilità legata all'input utente per generare un buffer overflow, e siccome tale approccio più si avvicina agli scopi dell'esercizio, abbiamo quindi deciso di seguire questa strada.

Si può facilmente notare che non ci sia alcun tipo di controllo sul tipo di input dell'utente, e questa è una grossa falla di questo codice.
Non essendoci controlli sul tipo di input, l'utente potrebbe inserire caratteri non numerici.
Questo potrebbe causare comportamenti anomali come un ciclo infinito causato dal fatto che, aspettandosi il programma un intero, e non riuscendo a processare come tale un carattere non numerico, continuerà a cercare di leggere quel dato creando un infinity loop.
D'altronde, questo è un problema già presente nello script, non una modifica che dobbiamo apportare noi, e comunque non è quello che ci serve per generare un buffer overflow.

Di per se il programma per come è scritto processa solo i primi 10 interi inseriti dall'utente, anche se quest'ultimo ne inserisse di più (per farlo all'utente basta scrivere più di 10 numeri separati da uno spazio e premere invio), e questo funge da controllo sul numero di input validi.
L'array infatti è definito in modo che ci siano 10 elementi al suo interno, e se quindi il programma processasse tramite il primo ciclo `for` più di questi elementi, ciò che si verificherebbe sarebbe che il programma comincerebbe a scrivere su celle di memoria non appartenenti all'array, causando un buffer overflow, ovvero ciò che stiamo cercando di fare.

La soluzione quindi è quella di modificare il primo ciclo `for` facendolo reiterare più di 10 volte, ovvero riscrivendolo ad esempio come `for (i=0 ; i<20 ; i++)`.

In realtà, per causare un errore di segmentazione serviranno probabilmente molte, ma molte, ulteriori iterazioni del ciclo, volendo esagerare lo faremo eseguire potenzialmente 1.000.000 di volte, e lo modificheremo quindi in `for (i=0 ; i<1000000 ; i++)`.

Abbiamo a questo punto pensato di modificare direttamente lo script includendo la parte banus. Abbiamo cercato di effettuare modifiche il più minimali possibili per alterare il meno possibile il funzionamento del programma originale.

Le implementazioni sono state:

- Inserire tutto il codice in un ciclo per non far arrestare il programma alla fine della sua esecuzione, facendo invece scegliere all'utente se continuare nel suo utilizzo o se arrestarlo
- Aggiungere un menu iniziale con la scelta se eseguire il codice "sano" o quello "malsano"
- Nella parte "sana", abbiamo inserito robusti controlli sull'input dell'utente, costruendo dei cicli per far si che l'errore dell'utente non blocchi il programma ma proponga all'utente di inserire input validi
- Nella parte "malsana" abbiamo aggiunto il comando che, se digitata una lettera e premuto invio, l'inserimento di interi si interrompe e il programma, se non si blocca, continua nella sua esecuzione

Il codice è molto lungo, viene qui incollato in formato testuale per maggiore chiarezza rispetto ad uno screeshot.

```
#include <stdio.h>

int main () {

    int scelta;
    int continua;

    // Ciclo principale per permettere la riesecuzione del programma
    do {
        // MENU DI SCELTA INIZIALE
        printf("\n=============== MENU ===============\n");
        printf("Scegli la modalita' di esecuzione:\n");
        printf("1. Esecuzione Sicura (con controllo input)\n");
        printf("2. Esecuzione con Buffer Overflow\n");
        printf("====================================\n");
        printf("Scelta: ");

        // Controllo sull'input della scelta
        while (scanf("%d", &scelta) != 1 || (scelta != 1 && scelta != 2)) {
            printf("Input non valido. Inserisci 1 o 2: ");
            while (getchar() != '\n'); // Pulisce il buffer da input errati
        }
        while (getchar() != '\n'); // Pulisce il buffer dal carattere 'a capo'

        // ESECUZIONE DELLA PARTE "SANA"
        if (scelta == 1) {
            int vector[10], i, j, k;
            int swap_var;

            printf("\n--- Modalita' Sicura ---\n");
            printf("Inserire 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. Reinserisci l'intero per [%d]:", c);
                    while (getchar() != '\n');
                }
            }

            printf("\nIl 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("\nIl vettore ordinato e':\n");
            for (j = 0; j < 10; j++) {
                int g = j + 1;
                printf("[%d]:", g);
                printf("%d\n", vector[j]);
```

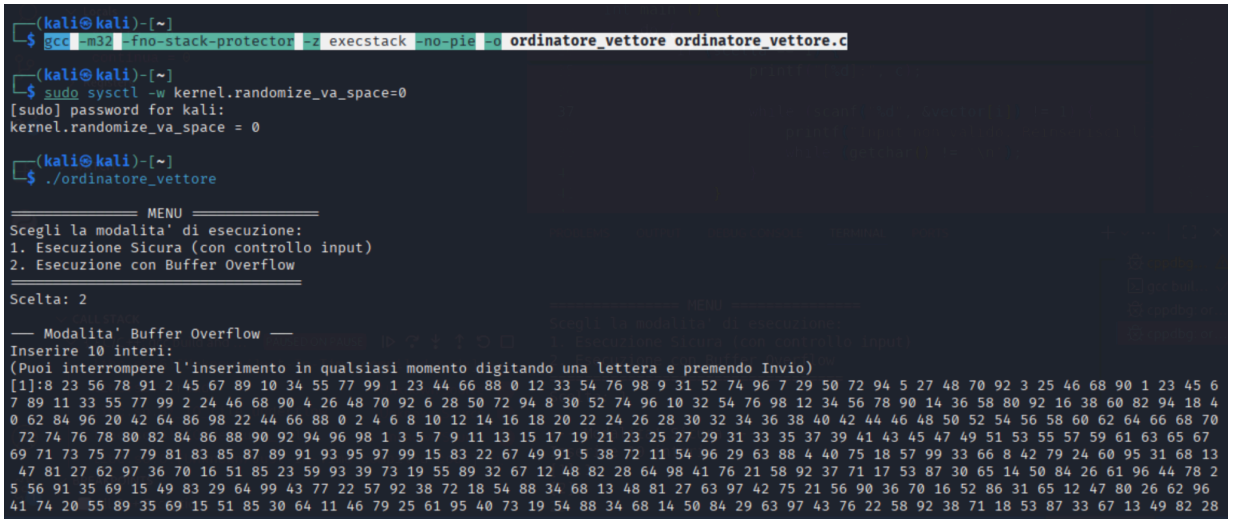
```
            }
        }
        // ESECUZIONE DELLA PARTE "MALSANA"
        else if (scelta == 2) {
            int vector[10], i, j, k;
            int swap_var;

            printf("\n--- Modalita' Buffer Overflow ---\n");
            printf("Inserire 10 interi:\n");

            // Messaggio aggiunto per l'utente
            printf("(Puoi interrompere l'inserimento in qualsiasi momento digitando una lettera e
premendo Invio)\n");

            for (i = 0; i < 1000000; i++) {
                int c = i + 1;
                printf("[%d]:", c);

                if (scanf("%d", &vector[i]) != 1) {
                    break;
                }
            }

            while (getchar() != '\n');

            printf("\nIl vettore inserito (se il programma non e' crashato) 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("\nIl vettore ordinato (se il programma non e' crashato) e':\n");
            for (j = 0; j < 10; j++) {
                int g = j + 1;
                printf("[%d]:", g);
                printf("%d\n", vector[j]);
            }
        }

        // COMANDO PER CONTINUARE O USCIRE
        printf("\nVuoi eseguire nuovamente il programma? (1 per Si, 0 per No): ");
        while (scanf("%d", &continua) != 1 || (continua != 0 && continua != 1)) {
            printf("Input non valido. Inserisci 1 per Si o 0 per No: ");
            while (getchar() != '\n');
        }
        while (getchar() != '\n');

    } while (continua == 1);

    printf("\nArrivederci!\n");

    return 0;
         }
```

Non ci resta quindi che eseguirlo e generare un errore di segmentazione.

Per farlo entriamo nel terminale Kali e rendiamo il nostro file ordinatore_vettore.c in un eseguibile in cui abbiamo disabilitato filtri e protezioni a livello di compilazione per rendere più semplice il presentarsi dell'errore desiderato con il comando `gcc -g -O0 -fno-stack-protector -z execstack -no-pie -o ordinatiore_vettore ordinatore_vettore.c`.
Analogamente abbiamo quindi disattivato ASLR, altra misura di protezione basata sulla randomizzazione dell'allocazione della memoria, da terminale con il comando `sudo sysctl -w kernel.randomize_va_space=0`.

Abbiamo quindi eseguito il programma e con l'inserimento di 20.002 interi abbiamo generato un errore di segmentazione.

Seguono screen a titolo dimostrativo.



…

…