



EPICODE



GODS OF HACKING  
ETHICAL HACKERS

# System Exploit BOF

L'obiettivo di oggi ci chiede di:

[https://drive.google.com/file/d/1nEM\\_FV5zFHj4hw9\\_Ya1PUP\\_xf5bLGy0I/view](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.

L'obiettivo come prima parte ci chiede di descrivere il funzionamento del programma in allegato prima dell'esecuzione, quindi per prima cosa apriamo l'allegato e iniziamo a visionarlo per capire di cosa si tratta, una volta fatto possiamo passare alla descrizione del funzionamento:

```
#include <stdio.h> //Include la libreria standard di input/output
int main () { //Definisce la funzione principale del programma
int vector [10], i, j, k; //Dichiara un array di 10 interi "vector" e tre variabili intere (i, j, k)
int swap_var; //Dichiara una variabile intera utilizzata per lo scambio di elementi nell'ordinamento.
printf ("Inserire 10 interi:\n"); //Visualizza un messaggio "Inserire 10 interi:"
for ( i = 0 ; i < 10 ; i++) //Inizia un ciclo che si ripeterà 10 volte
{
    int c= i+1; //Crea una variabile locale "c" che vale i+1
    printf("[%d]:", c); //Visualizza il numero dell'elemento corrente (da 1 a 10)
    scanf ("%d", &vector[i]); //Legge un intero inserito dall'utente e lo memorizza nell'elemento
i      i-esimo dell'array vector
}
printf ("Il vettore inserito e':\n"); //Visualizza un messaggio "Il vettore inserito e':"
for ( i = 0 ; i < 10 ; i++) //Scorre tutti i 10 elementi dell'array
{
    int t= i+1; //Crea una variabile locale t che vale i+1
    printf("[%d]: %d", t, vector[i]); //Mostra l'indice (da 1 a 10) e il valore dell'elemento
    printf("\n"); //Va a capo dopo ogni elemento
}
for (j = 0 ; j < 10 - 1; j++) //Ciclo esterno che si ripete 9 volte
{
    for (k = 0 ; k < 10 - j - 1; k++) //Ciclo interno che si ripete sempre - i volte ad ogni iterazione
{
```

```
if (vector[k] > vector[k+1]) //Verifica se l'elemento vector[k] è maggiore del
{
    successivo
    //Se la condizione è vera, scambia i due elementi usando la variabile
    temporanea swap_var
    swap_var=vector[k];
    vector[k]=vector[k+1];
    vector[k+1]=swap_var;
}
}

printf("Il vettore ordinato e':\n"); //Visualizza un messaggio "Il vettore ordinato e':"
for (j = 0; j < 10; j++) //Scorre tutti i 10 elementi dell'array
{
    int g = j+1; //Crea una variabile locale t che vale J+1
    printf("[%d]:", g); //Mostra l'indice (da 1 a 10)
    printf("%d\n", vector[j]); //Mostra il valore dell'elemento e va a capo
}
return 0;} //Termina il programma con codice di ritorno 0.
```

Il programma ordina un array di 10 elementi inseriti dall'utente e li ordina tramite algoritmo chiamato “bubble sort”. In questo report tenterò di sviscerarne il funzionamento, descrivendo i pezzi di codice e il funzionamento di quest'ultimi. Successivamente verificherò che le conclusioni siano corrette. Infine, modificherò il programma affinché si verifichi un errore di segmento. Come bonus, sempre modificando il codice, dovrò inserire un controllo sull'input e creare un menu per far decidere all'utente se utilizzare il codice che va in errore o quello corretto.

Analisi codice

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

Dichiarazione delle variabili per un algoritmo di ordinamento. 'vector' è un array di interi che conterrà 10 elementi da ordinare. Le variabili 'i', 'j' e 'k' serviranno come contatori nei cicli di iterazione durante il processo di ordinamento. La variabile 'swap\_var' verrà utilizzata come spazio temporaneo durante lo scambio di elementi dell'array nel processo di ordinamento

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

Ciclo for che acquisisce 10 valori interi dall'utente. Il ciclo itera da 0 a 9 e per ogni iterazione è creata una variabile 'c' che corrisponde alla posizione visualizzata all'utente (partendo da 1 anziché da 0); mastrandolo a schermo; salva il valore inserito dall'utente nella posizione corrispondente dell'array 'vector'.

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

Ciclo for che stampa i 10 valori interi inseriti dall'utente. Come prima Il ciclo iterà da 0 a 9 per mostrare a schermo l'indice; stampa il valore nella posizione corrispondente dell'array 'vector'.

```
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;
        }
    }
}
```

Implementazione del Bubble Sort che ordina l'array in modo crescente. Due cicli annidati: quello esterno (j) gestisce i passaggi completi, mentre quello interno (k) confronta e scambia elementi adiacenti quando necessario. Ad ogni passaggio esterno, i valori maggiori si spostano verso la fine dell'array, riducendo progressivamente l'area da ordinare.

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

in fine stampa l'array appena ordinato

La seconda parte dell'obiettivo ci chiede di riprodurre ed eseguire il programma nel laboratorio e di capire se le nostre ipotesi sul programma fossero corrette. Quindi eseguiamo il programma nel laboratorio e otteniamo questo che vediamo qui in figura:

```
Inserire 10 interi:  
[1]:9  
[2]:5  
[3]:6  
[4]:7  
[5]:2  
[6]:3  
[7]:0  
[8]:1  
[9]:8  
[10]:4  
Il vettore inserito e':  
[1]: 9  
[2]: 5  
[3]: 6  
[4]: 7  
[5]: 2  
[6]: 3  
[7]: 0  
[8]: 1  
[9]: 8  
[10]: 4  
Il vettore ordinato e':  
[1]:0  
[2]:1  
[3]:2  
[4]:3  
[5]:4  
[6]:5  
[7]:6  
[8]:7  
[9]:8  
[10]:9
```

Eseguendo il codice ci rendiamo conto che le supposizioni sono fondate. Il programma funziona esattamente come pronosticato, suddiviso in tre parti:

1. inserimento elementi nell'array,
2. ordinamento elementi nell'array,
3. stampa elementi ordinati dell'array.

La terza parte dell'obiettivo ci chiede di modificare il programma affinché si verifichi un errore di segmentazione:

```
#include <stdio.h>
int main () {
int vector [10], i, j, k;
int swap_var;
int num_elementi; // Variabile per controllare il numero di elementi
printf ("Quanti numeri vuoi inserire?");
scanf ("%d", &num_elementi); // Permettiamo all'utente di specificare quanti numeri inserire
printf ("Inserire %d interi:\n", num_elementi); // Non controlliamo se num_elementi > 10, causando potenzialmente un buffer overflow
for ( i = 0 ; i < num_elementi ; i++) //qui usiamo num_elementi anziché 10
{
    int c = i+1;
    printf("[%d]:", c);
    scanf ("%d", &vector[i]); // Potrebbe scrivere oltre i limiti dell'array
}
printf ("Il vettore inserito e':\n");
for ( i = 0 ; i < num_elementi ; i++) // Anche qui usiamo num_elementi anziché 10
{
    int t = i+1;
    printf("[%d]: %d", t, vector[i]);
    printf("\n");
}
```

```
for (j = 0 ; j < num_elementi - 1; j++) // Anche qui usiamo num_elementi anziché 10
{
    for (k = 0 ; k < num_elementi - j - 1; k++)// Anche qui usiamo num_elementi anziché 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 < num_elementi; j++) // Anche qui usiamo num_elementi anziché 10
{
    int g = j+1;
    printf("[%d]:", g);
    printf("%d\n", vector[j]);
}
return 0;}
```

La modifica che ho apportato può causare un errore di stack smashing detected, questo errore è generalmente correlato a un problema di danneggiamento dello stack, che può portare a errori di segmentazione. Questo errore viene generato quando il compilatore rileva che lo stack è stato compromesso, spesso a causa di un buffer overflow. L'exit code 134 indica che il programma è stato terminato forzatamente.

#### Differenze codice

```
int num_elementi;
printf ("Quanti numeri vuoi inserire? ");
scanf ("%d", &num_elementi);
```

Dichiarazione della variabile num\_elementi, usata per permettere all'utente di scegliere quanti elementi inserire. Successivamente, tutte le condizioni dei cicli for cambiano da 10 a num\_elementi.

```
*** stack smashing detected ***: terminated
...Program finished with exit code 134
```

L'obiettivo ha anche una parte bonus (facoltativa, ma in quanto masochisti di fama mondiale abbiamo completato anche questo) che ci richiede di inserire dei controlli input e creare un menù per far decidere all'utente se avere il programma che va in errore oppure quello corretto:

Inserire controlli input

```
int leggiIntero() {
    int numero;
    char buffer[256];
    while (1)
    {
        printf(": ");
        if (scanf("%d", &numero) == 1)
        {
            while (getchar() != '\n');
            return numero;
        } else {
            printf("Input non valido. Per favore inserisci solo numeri interi.\n");
            while (getchar() != '\n');
        }
    }
}= leggiIntero();
```

La funzione “leggiIntero()” è progettata per leggere un numero intero inserito dall'utente tramite la tastiera, assicurandosi che l'input fornito sia effettivamente un numero intero valido.

La condizione “getchar() != '\n'” in C serve a verificare se il carattere appena inserito dall'utente, tramite la funzione “getchar()”, non è un carattere di fine riga, cioè il carattere di nuova linea '\n'.

Funzione richiamata tramite “= leggiIntero();” ad esempio “numero = leggiIntero();” o “vettore[i] = leggiIntero();”

In conclusione, questa funzione permette di ottenere in modo affidabile un numero intero dall’utente, ripetendo la richiesta ogni volta che vengono inseriti valori non validi, fino a quando l’utente inserisce correttamente un numero intero.

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

```
int mostraMenu() {
    int scelta;
    printf("Scegli un'opzione:\n");
    printf("1. versione sana\n");
    printf("2. versione compromessa\n");
    scelta= leggiIntero();
    if (scelta == 1)
    {
        return 1;
    }
    else if (scelta == 2)
    {
        return 0;
    }
    else {
        printf("Scelta non valida. Per favore scegli tra 1 e 2.\n");
        return mostraMenu(); // Richiama la funzione nuovamente per permettere una seconda chance
    }
}
```

```
if (mostraMenu() == 0){  
    printf ("Quanti numeri vuoi inserire? ");  
    num_elementi= leggiIntero(); }  
else{  
    num_elementi= 10;  
}
```

La funzione “mostraMenu()” permette all’utente di scegliere se usare il codice compromesso o quello sano tramite una semplice condizione che restituisce 1 o 0 in base alla scelta. La funzione viene richiamata dal secondo pezzo di codice. Se la funzione restituisce 0, chiede: “Quanti numeri vuoi inserire?”, legge l’input e lo salva nella variabile num\_elementi. Se invece la funzione restituisce 1, alla variabile num\_elementi viene assegnato il valore 10.

```
#include <stdio.h>  
int num_elementi; // Variabile per controllare il numero di elementi  
int leggiIntero() {  
    int numero;  
    char buffer[256]; // Buffer per leggere la stringa input dell’utente  
    while (1) {  
        printf(": ");  
        if (scanf("%d", &numero) == 1) { // Controllo se ci sono altri caratteri nel buffer che non sono stati letti correttamente  
            while (getchar() != '\n');  
            return numero;  
        } else { // Se scanf non ha letto un intero, pulisci il buffer di input  
            printf("Input non valido. Per favore inserisci solo numeri interi.\n");  
            while (getchar() != '\n');  
        }}}
```

```
int mostraMenu() {
    int scelta;
    printf("Scegli un'opzione:\n");
    printf("1. versione sana\n");
    printf("2. versione compromessa\n");
    scelta= leggiIntero();
    if (scelta == 1) {
        return 1;
    } else if (scelta == 2) {
        return 0;
    } else {
        printf("Scelta non valida. Per favore scegli tra 1 e 2.\n");
        return mostraMenu(); // Richiama la funzione nuovamente per permettere una seconda chance
    }
}
int main () {
int vector [10], i, j, k;
int swap_var;
if (mostraMenu()==0){
    printf ("Quanti numeri vuoi inserire? ");
    num_elementi= leggiIntero(); }
else{
    num_elementi= 10; }
printf ("Inserire %d interi:\n", num_elementi); // Non controlliamo se num_elementi > 10, causando potenzialmente un buffer overflow
```

```
for ( i = 0 ; i < num_elementi ; i++){
    int c = i+1;
    printf("[%d]", c);
    vector[i]= leggiIntero(); // Potrebbe scrivere oltre i limiti dell'array
}
printf ("Il vettore inserito e':\n");
for ( i = 0 ; i < num_elementi ; i++) // Anche qui usiamo num_elementi anziché 10
{
    int t = i+1;
    printf("[%d] %d", t, vector[i]);
    printf("\n");}
for (j = 0 ; j < num_elementi - 1; j++) // Anche qui usiamo num_elementi anziché 10
{
    for (k = 0 ; k < num_elementi - j - 1; k++) // Anche qui usiamo num_elementi anziché 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 < num_elementi; j++) // Anche qui usiamo num_elementi anziché 10
{
    int g = j+1;
    printf("[%d]:", g);
    printf("%d\n", vector[j]);}
return 0;}
```

## CONSIDERAZIONI FINALI

### CONSIDERAZIONI FINALI:

- Il report si concentra sull'analisi e sulla modifica di un programma C per dimostrare una vulnerabilità di buffer overflow (BOF).
- Il documento guida il lettore attraverso il processo di comprensione del funzionamento del codice originale, che implementa un semplice algoritmo di ordinamento di un array.
- Successivamente, il report mostra come modificare il codice per introdurre una vulnerabilità di buffer overflow, consentendo all'utente di inserire più dati di quanti l'array possa contenere.
- Il report evidenzia l'importanza dei controlli sull'input dell'utente per prevenire vulnerabilità di sicurezza.
- Come bonus, il report descrive l'implementazione di un menu per consentire all'utente di scegliere se eseguire la versione vulnerabile o quella sicura del programma.

In conclusione, il report fornisce un esempio pratico di come si possa sfruttare una vulnerabilità di buffer overflow e sottolinea l'importanza di scrivere codice sicuro, con particolare attenzione alla validazione dell'input.