

Buffer overflow

Sommario

Traccia esercizio principale	2
Traccia esercizio facoltativo	4
Svolgimento esercizio	5
Creazione del codice.....	5
Creazione dell'eseguibile	5
Test sul programma con limite buffer 10	5
Modifica del limite buffer a 30	6
Risoluzione per evitare il Buffer Overflow	7
Implementazione del codice.....	8

Traccia esercizio principale

Il buffer overflow è una vulnerabilità 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).

Riportate il codice che segue sulla vostra Kali Linux, creando un nuovo documento con estensione .c sul desktop.

```
GNU nano 6.3
#include <stdio.h>

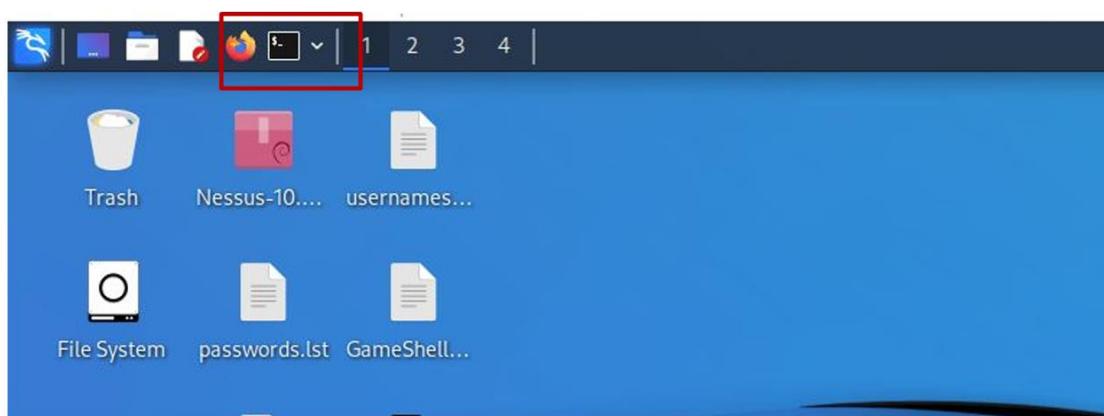
int main () {
    char buffer [10];

    printf ("Si prega di inserire il nome utente:");
    scanf ("%s", buffer);

    printf ("Nome utente inserito: %s\n", buffer);

    return 0;
}
```

Per creare un nuovo documento su Kali, avviate la vostra Kali Linux, ed una volta presentata la schermata principale cliccate sull'icona del terminale mostrata nel rettangolo rosso in figura.



Dal terminale, poi, spostatevi sul desktop eseguendo il comando: **cd /home/Kali/Desktop**

```
[kali㉿kali)-[~]
File Actions Edit View Help
└─$ cd /home/kali/Desktop
```

Successivamente, eseguiamo l'editor di testo nano, che ci permette o di aprire un file esistente oppure di crearne uno nuovo se il nome del file specificato non esiste. Eseguiamo quindi dal terminale il comando di seguito per creare un file BOF.c

nano BOF.c

Riportate il frammento di codice nella figura a destra nel file che avete appena creato, facendo attenzione a riportare tutti i simboli, potete modificare a vostro piacimento il contenuto delle «printf» tra le virgolette.

```
#include <stdio.h>

int main () {
    char buffer [10];

    printf ("Si prega di inserire il nome utente:");
    scanf ("%s", buffer);

    printf ("Nome utente inserito: %s\n", buffer);

    return 0;
}
```

Una volta completato, chiudete e salvate il file. Per chiudere e salvare un file con l'editor di testo nano, dovete seguire una sequenza di comandi da tastiera:

1. Premete insieme i tasti Ctrl e x sulla vostra tastiera
2. Il tool vi chiederà se volete salvare il programma. Digitate «y» e poi premete «invio» per salvare il file con il nome scelto

A questo punto, compilate il file utilizzando il comando **gcc -g BOF.c -o BOF**

```
[kali㉿kali)-[~/Desktop]
$ gcc -g BOF.c -o BOF
```

Ricordate di compilare il programma ad ogni modifica del codice

Una volta fatto, potete eseguire il programma eseguendo il comando **./BOF**

```
[kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:■
```

Il programma si avvia chiedendoci di inserire un nome utente:

- Inserendo un nome utente di 5 caratteri, il programma non ci riporta nessun problema, infatti come sappiamo il buffer accetta fino a 10 caratteri. Cosa succede se inseriamo 30 caratteri? **Proviamo**

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:test1
Nome utente inserito: test1

(kali㉿kali)-[~/Desktop]
$
```

```
Si prega di inserire il nome utente:qwertyuiopasdfghjklzxcvbnmqwer
Nome utente inserito: qwertyuiopasdfghjklzxcvbnmqwer
zsh: segmentation fault  ./BOF

(kali㉿kali)-[~/Desktop]
$
```

Se inseriamo 30 caratteri il programma ci ritorna un errore, «segmentation fault», ovvero errore di segmentazione. L'errore di segmentazione avviene quando un programma, come abbiamo detto in precedenza, tenta di scrivere contenuti su una porzione di memoria alla quale non ha accesso. Questo è un chiaro esempio di BOF, abbiamo inserito 30 caratteri in un buffer che ne può contenere solamente 10 e di conseguenza alcuni caratteri stanno sovrascrivendo aree di memorie inaccessibili.

Provate a riprodurre l'errore di segmentazione modificando il programma come di seguito:

- Aumentando la dimensione del vettore a 30.

Questo basta ad eliminare la possibilità di generare un BOF?

Traccia esercizio facoltativo

Aggiungete nel programma precedente dei controlli di sicurezza per prevenire il BOF e sanitizzare l'input.

Svolgimento esercizio

Creazione del codice

```
home > kali > Desktop > BOF.c
1 #include <stdio.h>
2
3 int main () {
4
5     char buffer [10];
6     printf("Si prega di inserire il nome utente:");
7     scanf ("%s",buffer);
8
9     printf ("Nome utente inserito: %s\n", buffer);
10
11 }
12
```

Creazione dell'eseguibile

```
(kali㉿kali)-[~/Desktop]
$ gcc -g BOF.c -o BOF

(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:
```

Utilizzare il comando di terminale, dalla stessa cartella del file BOF.c: **gcc -g BOF.c -o BOF** per compilare il file sorgente BOF.c, includendo informazioni di debug nell'eseguibile, e generare un file eseguibile chiamato BOF. In questo modo, sarà possibile eseguire il programma con il comando **./BOF**.

Test sul programma con limite buffer 10

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:ciao
Nome utente inserito: ciao

(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:qwiжdddddqwieuoiqwuyeoiqwueioqwuiеouqw
Nome utente inserito: qwiжdddddqwieuoiqwuyeoiqwueioqwuiеouqw
zsh: segmentation fault  ./BOF

(kali㉿kali)-[~/Desktop]
$
```

Come da traccia, il nome utente **ciao** è riconosciuto, mentre superando i 10 caratteri il programma da errore.

Modifica del limite buffer a 30

```
home > kali > Desktop > C BOF.c
1 #include <stdio.h>
2
3 int main () {
4     char buffer [30];
5     printf("Si prega di inserire il nome utente:");
6     scanf ("%s",buffer);
7
8     printf ("Nome utente inserito: %s\n", buffer);
9
10    return 0;
11 }
12 }
```



Si ripete la compilazione del programma: **gcc -g BOF.c -o BOF**

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:ciao ciao si i si i si i si i si i si i
Nome utente inserito: ciao ciao si i si i si i si i si i si i

(kali㉿kali)-[~/Desktop]
```

Questo aumenta il limite solo a 30 caratteri, superato questo limite si ottiene lo stesso errore, pertanto non risolve il problema.

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:whjequheiuiwhuihruiwqhreuowqjoidjoiqwjd
jeioqw
Nome utente inserito: whjequheiuiwhuihruiwqhreuowqjoidjoiqwjd
zsh: segmentation fault ./BOF
```

Risoluzione per evitare il Buffer Overflow

In C, le stringhe sono rappresentate come array di caratteri terminati dal carattere speciale \0, noto come null terminator o terminatore di stringa. Questo carattere \0 viene aggiunto automaticamente alla fine di ogni stringa per indicare dove la stringa termina. Questo significa che il buffer può contenere al massimo 9 caratteri validi più 1 carattere riservato al terminatore di stringa \0. Infatti, lo spazio totale del buffer è 10, ma l'ultimo elemento dell'array deve essere riservato per il carattere \0.

```
 kali@kali:~/Desktop$ ./BOF.c
#include <stdio.h>

int main () {
    char buffer [10];
    printf ("Si prega di inserire il nome utente:");
    scanf ("%9s", buffer);
    printf ("Nome utente inserito: %s\n", buffer);
    return 0;
}
```

Per evitare il buffer overflow con scanf, bisogna limitare il numero di caratteri che scanf può leggere. Questo si fa specificando un limite nel formato della stringa di input. In questo caso, con un buffer di 10 caratteri, si può fare in modo che scanf legga al massimo 2 caratteri (lasciando spazio per il terminatore di stringa \0).

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:jsdijasjjsdij
Nome utente inserito: jsdijasij
```

Implementazione del codice

```
home > kali > Desktop > C BOF_implementato.c
 1 #include <stdio.h>
 2 #include <string.h>
 3
 4 int main() {
 5
 6     char buffer[15]; // Buffer per il nome utente (14 caratteri + '\0')
 7
 8     // Ciclo che continua finché l'utente non inserisce un nome valido
 9     while (1) {
10         printf("Si prega di inserire il nome utente (massimo 10 caratteri): ");
11
12         // Legge massimo 14 caratteri dall'input
13         scanf("%14s", buffer);
14
15         // Svuota il buffer di input per rimuovere eventuali caratteri in eccesso
16         int c;
17         while ((c = getchar()) != '\n' && c != EOF); // Svuota il buffer fino al newline
18
19         // Verifica la lunghezza dell'input
20         if (strlen(buffer) < 11) {
21             // Se la lunghezza è valida (meno di 11 caratteri), esce dal ciclo
22             break;
23         } else {
24             // Se l'input è troppo lungo, stampa un messaggio di errore
25             printf("Nome utente non valido, si prega di riprovare.\n");
26         }
27     }
28
29     // Stampa il nome utente inserito
30     printf("Nome utente inserito: %s\n", buffer);
31
32     return 0;
33 }
```

Per evitare un buffer overflow con `scanf` (riga 11), l'input è limitato a 14 caratteri, lasciando spazio per il carattere terminatore `\0` in un buffer di 15 caratteri. Anche se l'utente viene invitato a inserire massimo 10 caratteri (riga 10), il limite di 14 garantisce sicurezza e previene l'overflow.

Il ciclo `while` verifica che l'utente inserisca un nome con meno di 11 caratteri (10 caratteri + `\0`). Se l'input è valido, il ciclo si interrompe con `break` (riga 16) e il programma stampa il nome utente (riga 24). Se l'input è troppo lungo, viene mostrato un messaggio di errore e si richiede un nuovo inserimento.

Poiché `scanf` non gestisce i caratteri in eccesso lasciati nel buffer, è stato aggiunto un ciclo `while` con `getchar()` per svuotare il buffer. `getchar()` legge un carattere alla volta dall'input fino a trovare un newline (`\n`) o EOF (End Of File).

- `c = getchar()`: Legge un carattere e lo assegna a `c`.
- `c != '\n'`: Il ciclo continua finché non trova un newline.
- `c != EOF`: Se raggiunge la fine dell'input, il ciclo termina.

Questo assicura che, quando il ciclo richiede un nuovo input, il buffer sia vuoto, evitando che i caratteri residui interferiscano con il prossimo inserimento.

Senza svuotare il buffer, il codice terminerebbe così:

```
[kali㉿kali)-[~/Desktop] $ ./BOF_plus
Si prega di inserire il nome utente (massimo 10 caratteri): jdsialjdlisajlidsjlilij
Nome utente non valido, si prega di riprovare.
Si prega di inserire il nome utente (massimo 10 caratteri): Nome utente inserito: dsjlijlij
```

Codice correttamente implementato:

```
[kali㉿kali)-[~/Desktop] $ gcc -g BOF_implementato.c -o BOF_plus
[kali㉿kali)-[~/Desktop] $ ./BOF_plus
Si prega di inserire il nome utente (massimo 10 caratteri): jdiwjqoiwjeoiqjeoiwqjeojij
Nome utente non valido, si prega di riprovare.
Si prega di inserire il nome utente (massimo 10 caratteri): sdjjd
Nome utente inserito: sdjjd
```