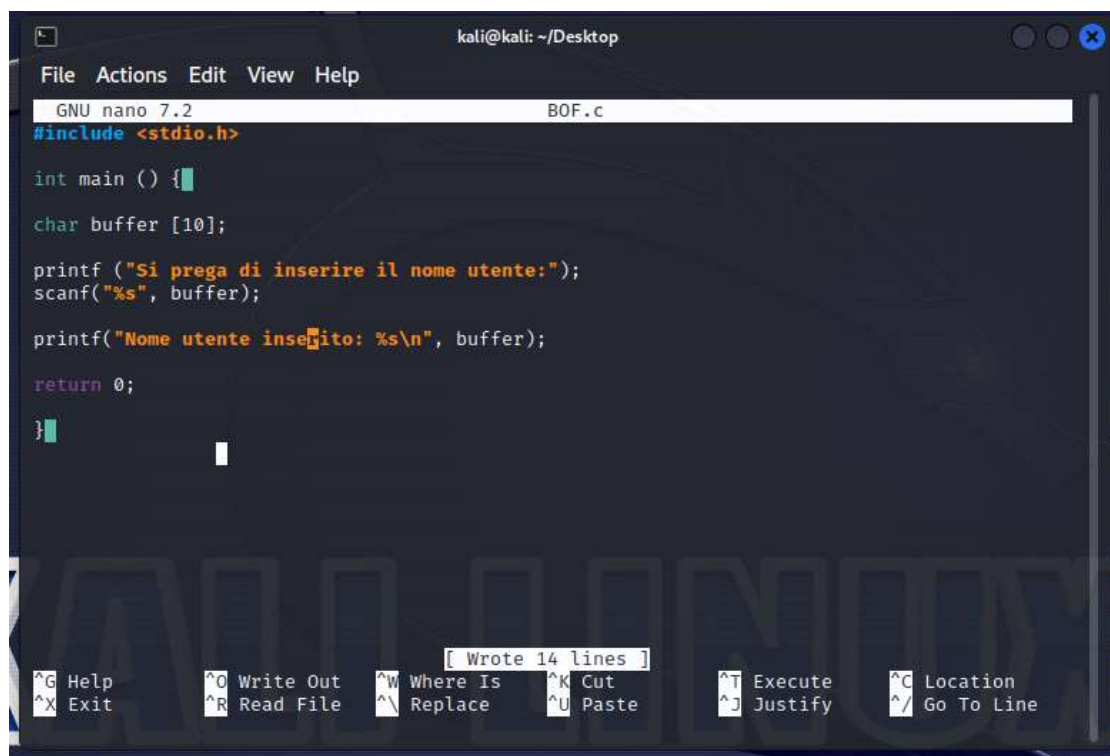


Buffer overflow

Durante esercizio di oggi andremo a capire cos'è e come funziona il buffer overflow.

Il buffer overflow è una vulnerabilità di sicurezza informatica che si verifica quando un programma, durante la scrittura di dati in un buffer, supera i limiti di memoria assegnati a tale buffer. Ciò può causare la sovrascrittura della memoria adiacente, incluso lo spazio di memoria che contiene altri dati importanti. Può essere causato da input non validati o non controllati che superano la dimensione del buffer previsto.

Questo overflow può essere sfruttato dai malintenzionati per iniettare codici malevoli oppure per far crashare le memorie buffer inserendo molti valori che si sovrascrivono.



```
kali@kali: ~/Desktop
File Actions Edit View Help
GNU nano 7.2 BOF.c
#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;
}

[ Wrote 14 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Questo è il programma di partenza dove abbiamo un limite di 10 caratteri nella variabile char buffer, dato che questa variabile non è sanata noi, inserendo più caratteri possiamo vedere come possiamo fruttare il buffer overflow; inserendo più caratteri il programma va in errore dicendo zsh: segmentation fault ./BOF quindi siamo andati a sovrascrivere i dati in eccesso sopra ad altri.

Una prima cosa che si può fare per ovviare a questo problema è quella di aumentare il valore dei caratteri che è possibile inserire in output, per esempio a 30.

```
kali@kali: ~/Desktop
File Actions Edit View Help
GNU nano 7.2 BOF.c
#include <stdio.h>

int main () {
char buffer [30];

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

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

return 0;
}

[ Read 14 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Però questo non è la soluzione definitiva dato che l'utente può creare problema inserendo ad esempio 50 caratteri.

```
kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:123654987123654897123654987132
Nome utente inserito: 123654987123654897123654987132

(kali@kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:213456789132564789123456789123645789123645978123654978
Nome utente inserito: 213456789132564789123456789123645789123645978123654978
zsh: segmentation fault ./BOF

(kali@kali)-[~/Desktop]
$
```

Quindi diciamo che la soluzione sarebbe quella di andare a sanare, nel codice, l'input dell'utente limitandolo alla quantità di caratteri che noi desideriamo.

Ad esempio utilizzare 'fgets' invece di 'scanf' ad esempio:

```
kali@kali: ~/Desktop
File Actions Edit View Help
GNU nano 7.2 BOF.c
#include <stdio.h>

int main () {
char buffer [30];

printf ("Si prega di inserire il nome utente:");
fgets(buffer, sizeof(buffer), stdin);

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

return 0;
}

[ Wrote 14 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

Il programma in questo caso prenderà un massimo di 30 caratteri e non di più

evitando il buffer overflow.