

Report sull'esercizio relativo ai Buffer Overflow

Data 18/01/23

Introduzione:

Nell'ambito delle lezioni sugli attacchi di sistema, è stato affrontato il tema dei buffer overflow (BOF), una vulnerabilità derivante dalla mancanza di controllo sui limiti dei buffer che accettano input utente. L'obiettivo dell'esercizio è comprendere i rischi associati ai BOF e sperimentare un caso pratico attraverso la modifica di un programma in linguaggio C.

Obiettivo dell'esercizio:

L'obiettivo principale dell'esercizio è studiare e comprendere i buffer overflow attraverso un esempio di codice in C volutamente vulnerabile come da screen:

```
#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;
}
```

Svolgimento:

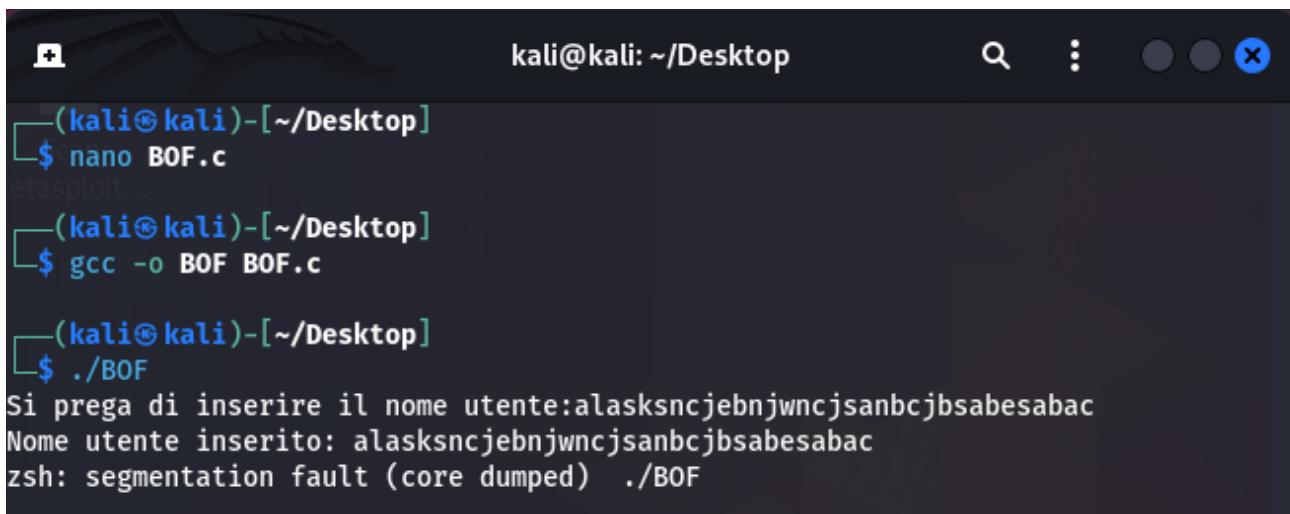
Compilazione del codice:

Riportato il codice fornito attraverso la compilazione del codice utilizzando il comando gcc con l'opzione -g abilitiamo le informazioni di debug per facilitare

l'analisi del codice. Eseguiamo il programma tramite il comando `./BOF`. Il programma richiede l'inserimento di un nome utente.

Scatenare un Buffer Overflow:

Se inseriamo un nome utente di 30 caratteri, oltre il limite di 10 previsto dal programma, il programma genera un "segmentation fault", un errore di memoria che si verifica quando il programma tenta di scrivere in una posizione di memoria non consentita.



```
kali@kali: ~/Desktop
(kali@kali)-[~/Desktop]
$ nano BOF.c
(kali@kali)-[~/Desktop]
$ gcc -o BOF BOF.c
(kali@kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:alasksnjcjebnjwncjsanbcjbsabesabac
Nome utente inserito: alasksnjcjebnjwncjsanbcjbsabesabac
zsh: segmentation fault (core dumped) ./BOF
```

Modifiche al Codice:

Abbiamo effettuato diverse modifiche al codice originale per illustrare la gestione dei buffer overflow e implementare correzioni atte a prevenire questi problemi di sicurezza.

Inizialmente, abbiamo aumentato la dimensione dell'array buffer da 10 a 30 caratteri per consentire l'inserimento di una stringa più lunga. Questa modifica è stata effettuata al fine di creare condizioni che non portino a un buffer overflow.

Successivamente, abbiamo introdotto un ciclo `do while` per gestire le condizioni relative alla lunghezza dell'input inserito dall'utente. Il ciclo permette di continuare a richiedere l'input finché non viene inserita una stringa con una lunghezza inferiore o uguale a 30 caratteri.

All'interno del ciclo, abbiamo utilizzato la funzione `fgets` al posto di `scanf` per gestire l'input, migliorando la robustezza del programma.

La scelta di sostituire scanf con fgets è stata dettata dalla maggiore sicurezza offerta da fgets nella gestione degli input utente. fgets consente di specificare la dimensione massima dell'input, riducendo il rischio di buffer overflow.

Utilizzando fgets puoi leggere l'intera linea di input (inclusi spazi e caratteri di nuova riga) fino a quando viene raggiunta la dimensione massima del buffer. Questo ti consente di gestire meglio situazioni in cui l'utente inserisce più di 30 caratteri.

Tuttavia, abbiamo rilevato che l'implementazione iniziale di fgets aveva un problema relativo ai caratteri in overflow, poiché venivano considerati come input per il nuovo ciclo. Quindi abbiamo inserito due righe di codice per evitare questi problemi ovvero pulire il buffer dell'input precedente, così da evitare che i caratteri di overflow del primo input venivano presi in automatico dal programma come input del nuovo ciclo di richiesta del nome utente.

Di seguito gli screen del codice modificato e della risposta del programma:

```
int main() {

char buffer[31];

do {

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

    // Pulisce il buffer di input
    int c;
    while ((c = getchar()) != '\n' && c != EOF);

    // Rimuove il carattere di nuova linea finale inserito da fgets
    buffer[strcspn(buffer, "\n")] = '\0';

    if (strlen(buffer) ≥ 30) {
        printf("Puoi inserire massimo 30 caratteri\n");
    } else {
        printf("Nome utente inserito: %s\n", buffer);
    }

} while (strlen(buffer) ≥ 30);

return 0;

}
```

```
kali@kali: ~/Desktop

(kali@kali)-[~/Desktop]
$ rm BOF

(kali@kali)-[~/Desktop]
$ gcc -o BOF BOF.c

(kali@kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:
123456789012345678901234567890mmm
Puoi inserire massimo 30 caratteri
Si prega di inserire il nome utente:
█
```

Analisi degli indirizzi di memoria

Abbiamo eseguito diverse modifiche per analizzare il comportamento del programma quando vengono inseriti più di 30 caratteri, andando oltre la dimensione massima del buffer.

Per eseguire la verifica degli indirizzi di memoria e vedere dove vengono salvati i caratteri di overflow, abbiamo introdotto istruzioni di stampa aggiuntive nel blocco di codice che gestisce il buffer overflow. Nello specifico:

- `printf("Caratteri in overflow: %s\n", buffer + 30);`
- `printf("Posizione in memoria dei caratteri di overflow: %p\n", (void*)(buffer + 30));`

Gli screen mostrano l'aggiunta di queste stringhe di codice e il risultato dell'esecuzione del programma

```

#include <stdio.h>
#include <string.h>

int main() {

char buffer[31];

do {
    // Ci dice dove viene memorizzato la nostra variabile buffer (serve da confronto)
    printf("Indirizzo di memoria del buffer: %p\n", (void*)buffer);

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

    // Pulisce il buffer di input
    int c;
    while ((c = getchar()) != '\n' && c != EOF);

    // Rimuove il carattere di nuova linea finale inserito da fgets
    buffer[strcspn(buffer, "\n")] = '\0';

    if (strlen(buffer) >= 30) {
        printf("Puoi inserire massimo 30 caratteri\n");
        printf("Caratteri in overflow: %s\n", buffer + 30);
        printf("Posizione in memoria dei caratteri di overflow: %p\n", (void*)(buffer + 30));
    } else {
        printf("Nome utente inserito: %s\n", buffer);
    }

} while (strlen(buffer) >= 30);

return 0;

}

```

```

(kali@kali)-[~/Desktop]
└─$ ./BOF
Indirizzo di memoria del buffer: 0x7fffe1b752a0
Si prega di inserire il nome utente:
123456789012345678901234567890mmm
Puoi inserire massimo 30 caratteri
Caratteri in overflow:
Posizione in memoria dei caratteri di overflow: 0x7fffe1b752be
Indirizzo di memoria del buffer: 0x7fffe1b752a0
Si prega di inserire il nome utente:

```

Conclusioni:

Queste modifiche al codice ci consentono di osservare direttamente il comportamento del programma quando si verifica un buffer overflow. Verificando la posizione in memoria e i caratteri coinvolti, acquisiamo una maggiore comprensione del rischio associato ai BOF. Questa analisi è cruciale per l'implementazione di misure preventive e la scrittura di codice più robusto, riducendo le potenziali vulnerabilità di sicurezza.

