Vulnerabilità di buffer overflow ed attacchi

Principalmente le vulnerabilità di buffer overflow più conosciute sono le seguenti:

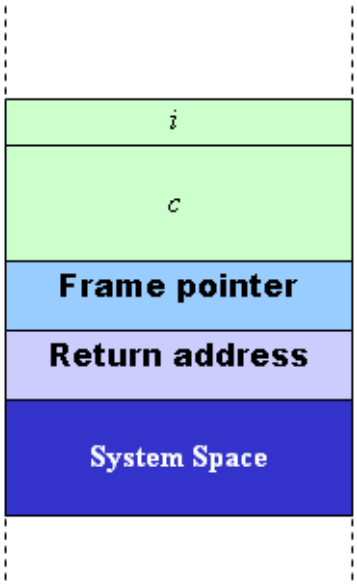
- *classico*
- *frame pointer*
- *heap*

Buffer overflow classico

Il buffer overflow *classico*, ideato da Aleph1 nell'articolo "Smashing the stack for fun and profit" nella e-zine Phrack #49, è il padre di tutti i buffer overflow. Un esempio di questo tipo di buffer overflow è dato di seguito:

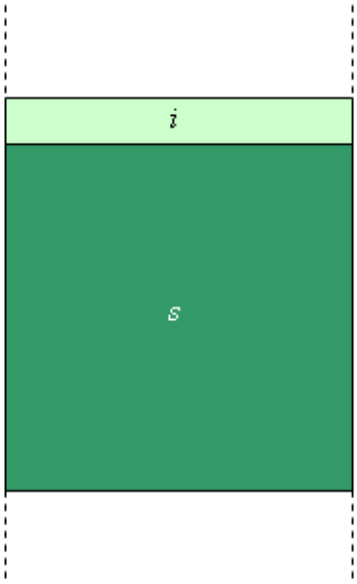
```
int    f(char   *s)

{
      char c[16];
      int i;

      i = 1;
      strcpy (c, s);

      return i;
}
```

Nel codice esempio dato sopra la funzione *strcpy* copia i dati da **s** a **c** senza effettuare alcun controllo sulla dimensione del buffer di destinazione (**c** appunto): è proprio questa la vulnerabilità del nostro programma. Se la stringa passata in argomento (ovvero **s**) dovesse presentare una dimensione superiore a 16 byte, si verificherebbe il tanto paventato buffer overflow perché il buffer **c** non è in grado di contenere tutti i dati e quindi si finirebbe con l'andare a sovrascrivere una zona di memoria non prevista.
Inizialmente lo stack frame della funzione *f* può essere rappresentato come segue:



In seguito ad una chiamata alla funzione *f*, la zona di memoria allocata alla variabile **c** dovrebbe essere sovrascritta dai dati contenuti in **s**. Tuttavia, se **s** è abbastanza lunga, potrebbe andare a sovrascrivere anche **frame pointer**,

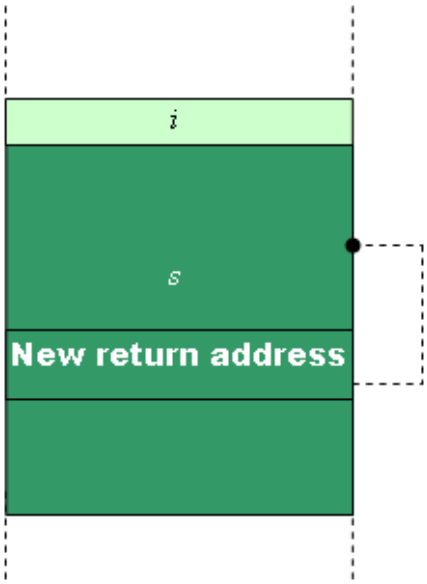
return address e altro ancora e portare ad una situazione simile alla seguente:



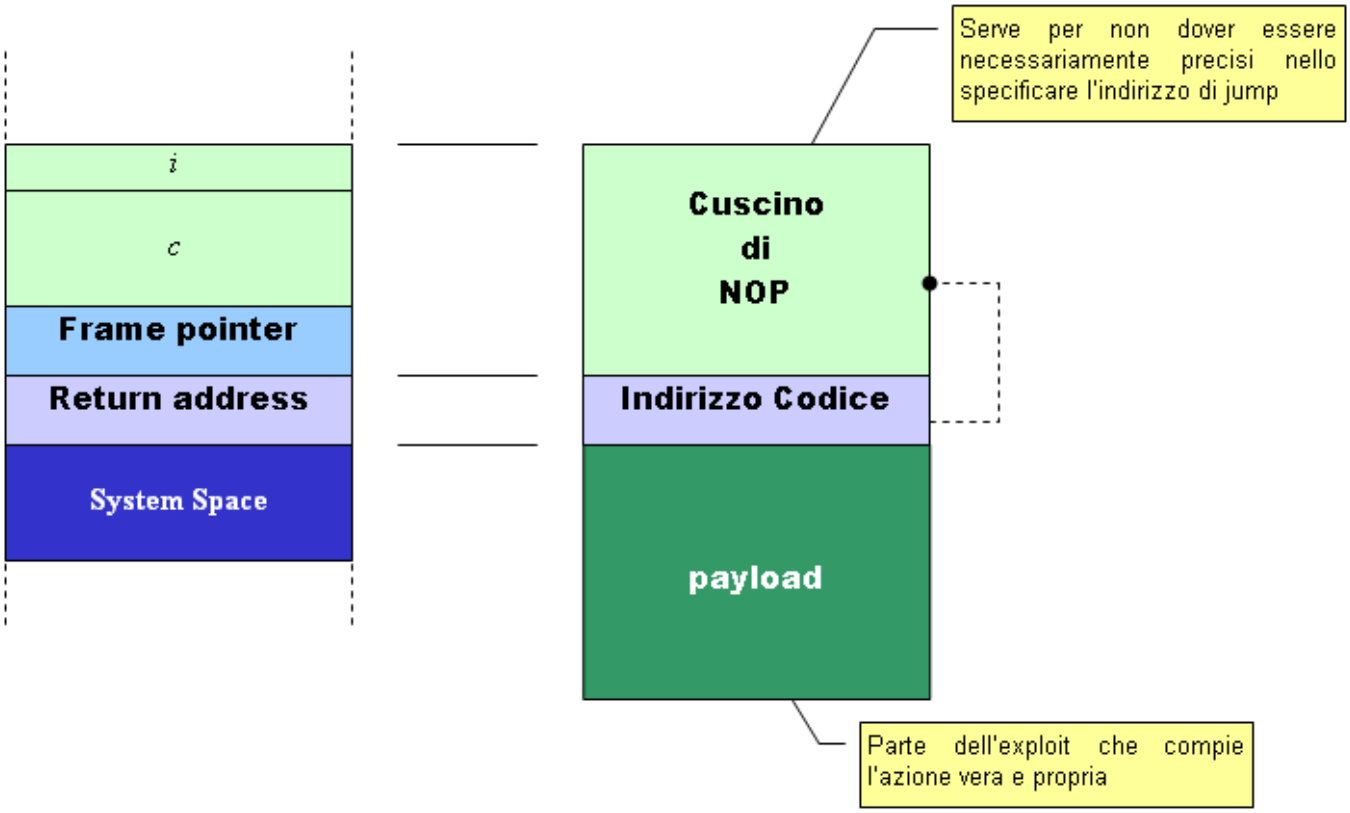
La stringa **s** passata come argomento alla funzione può essere costruita ad hoc in modo tale da:

- contenere delle istruzioni che verranno fatte eseguire all'host su cui gira il programma
- andare a sovrascrivere il **return address** affinché punti all'interno dei dati iniettati usando la stringa **s**

Così facendo, quando *f* termina la propria esecuzione, anziché ritornare all'istruzione successiva alla chiamata alla funzione, il nuovo valore contenuto in **return address** fa saltare l'esecuzione all'interno dei dati iniettati. Di seguito mostriamo una simile eventualità:



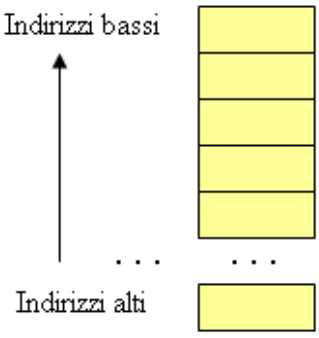
Usando termini più propriamente relativi all'ambito degli exploit, si ha quanto segue:



Mostriamo, ora, un altro esempio di buffer overflow classico in cui supporremo di lavorare in un'architettura semplificata con la word di un byte. Supponiamo di avere la seguente funzione:

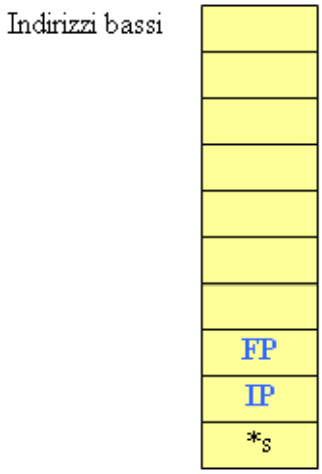
```
...
f("ciao");
...

void f (char *s)
{
     char b[4];
     strcpy (b, s);
}
```

Lo stack, prima dell'invocazione della funzione *f*, si trova nel seguente stato:

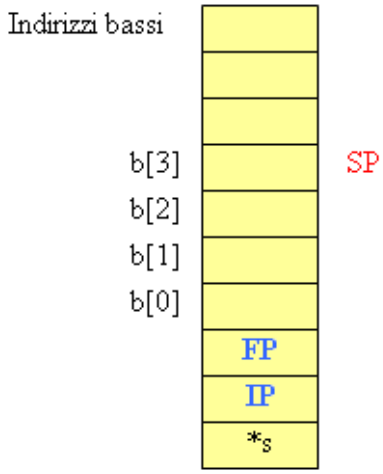


Questo codice non fa altro che copiare la stringa di caratteri "ciao" (parametro attuale passato alla funzione) nel buffer di caratteri b allocato sullo stack. Quando viene invocata la funzione *f* viene allocata memoria per i parametri

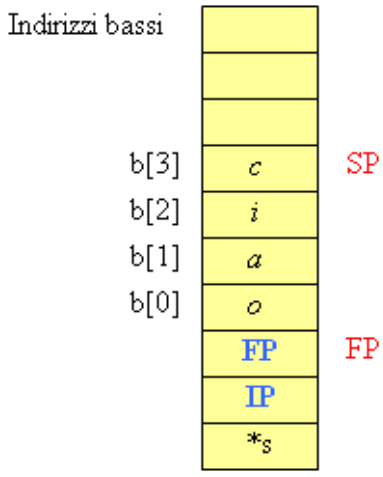
nello stack, vengono salvati nello stack l'IP (Instruction Pointer), l'FP e viene creato il nuovo SP:
Dopo l'invocazione della funzione *f*, lo stack appare come segue:



A questo punto viene decrementato SP per allocare spazio per l'array b[4]:



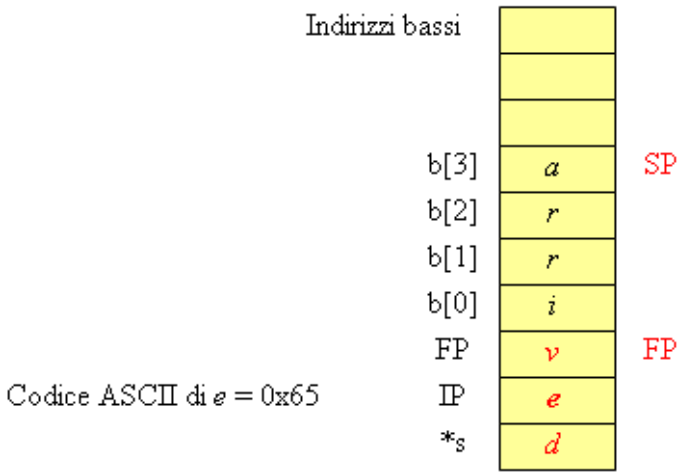
Subito dopo l'esecuzione dell'istruzione strcpy(b,s) la situazione nello stack sarà la seguente:



All'uscita dalla funzione chiamata vengono recuperati il Frame Pointer (FP) e l'Instruction Pointer (IP) dallo stack e ripristinati nei rispettivi registri in modo da far proseguire l'esecuzione del programma principale con l'istruzione

successiva alla chiamata di *f*.

Vediamo adesso un caso in cui viene volutamente provocato un overflow del buffer sullo stack, inserendo una stringa più lunga di quanto consentito dalle dimensioni dell'array (il buffer) allocato. Proviamo ad inserire la stringa "arrivederci":



Siccome la funzione non prevede alcun controllo sulla dimensione del parametro passato, la stringa ("arrivederci") è stata accettata nonostante la sua lunghezza (11) fosse maggiore della capacità del buffer (4). Questo ne provoca l'overflow e la conseguente sovrascrittura del FP, IP ed il puntatore *s. All'uscita dalla funzione quindi, poiché l'IP non conterrà più il corretto valore di ritorno, ma 0x65 (che altro non è che il codice ASCII della lettera "e") il flusso d'esecuzione del programma verrà deviato proprio verso questo nuovo indirizzo che non era affatto quello previsto.

Una volta che la funzione chiamata termina, il processore tenterà di eseguire l'istruzione contenuta all'indirizzo indicato dall'IP, incorrendo pertanto in uno di questi casi:

- il numero esadecimale finito nell'IP rappresenta un indirizzo che va fuori dall'intero spazio di memoria dedicato al processo. In tal caso si genera un segmentation fault;
- il numero esadecimale finito nell'IP rappresenta un indirizzo valido per il processo in esecuzione. In tal caso si ha un malfunzionamento del programma.

A questo punto è chiaro che grazie a questa semplice mancanza di cura nella programmazione si può alterare il normale flusso d'esecuzione del programma. Ciò significa che è possibile riferirsi ad un indirizzo di memoria scelto che permetta di eseguire codice arbitrario voluto dall'attaccante. In tal modo si riesce ad avere l'accesso ed il controllo di una macchina senza essere in possesso di nessuna shell o account sulla stessa, ovvero si ottiene quello che viene chiamato un exploit da remoto. La cosa però non è così semplice come sembra. Anzitutto è necessario trovare un posto adeguato dove scrivere il codice voluto. Poiché la sezione testo del processo è a sola lettura, il posto più adeguato risulta proprio il buffer stesso, fermo restando che il buffer abbia dimensione sufficiente a contenere il codice (altrimenti bisogna ricorrere ad altre tecniche più avanzate per reindirizzare l'esecuzione in punti diversi). L'altro rilevante problema sta nel riuscire ad indirizzare l'IP all'inizio del buffer perché è proprio a partire da lì che l'attaccante vorrà posizionare il suo programma. Bisognerà quindi pensare principalmente a realizzare la stringa da copiare nel buffer in modo che contenga nella parte iniziale il codice del programma da "exploitare" ed in seguito l'indirizzo di allocazione del buffer che dovrà andare a sovrascrivere l'IP.