C/C++ E OTTIMIZZAZIONE DEL CODICE
evilsocket
http://www.evilsocket.net/

.: Introduzione

Una delle problematiche più importanti che si va ad affrontare quando si scrive codice C/C++ a livelli avanzati è l'ottimizzazione .
Tale processo consiste nel rivedere i propri algoritmi e, in base a piccoli accorgimenti minimizzarne il tempo di esecuzione, allo scopo di rendere tutto il programma più veloce .
Naturalmente più una determinata funzione viene richiamata nel codice, più dovrà essere ottimizzata, quindi mi raccomando, prestate attenzione alle funzioni che vengono usate spesso ;) .

Andiamo ora a vedere alcuni esempi di ottmizzazione, tanto per capire di cosa stiamo parlando .

CODICE SBAGLIATO :

```
char stringa[0xFF];
int  i;

for( i = 0; i < strlen(stringa); i++ ) ...
```

CODICE OTTIMIZZATO :

```
char stringa[0xFF];
int  i,
     len = strlen(stringa);

for( i = 0; i < len; i++ ) ...
```

Nel primo esempio, ogni volta che viene eseguito il ciclo for, il sistema controlla la condizione di uscita eseguendo ad ogni loop la strlen, cosa inutile dato che restituirà sempre lo stesso valore, quindi conviene dichiarare una variabile len come nel secondo blocco di codice ed eseguire il controllo con quella (la stessa cosa vale anche per gli altri costrutti iterativi ovviamente) .

CODICE SBAGLIATO :

```
int n = 0;

while(true)
{
  ... operazioni ...

  if( n++ > 10 )
      break;

}
```

CODICE OTTIMIZZATO :

```
do
{
  ... operazioni ...
}
while( n++ <= 10 );
```

Altro esempio su un ciclo iterativo, in questo caso venivano fatti due controlli per la condizione di uscita, il primo nel while(true) e poi nel if ... il tutto è equivalente ad un solo controllo come nel secondo blocco, guadagnando in tempo di esecuzione .

Come vedete bastano dei piccoli accorgimenti nella struttura dei propri algoritmi, in più ci vengono in aiuto alcune caratteristiche del C e del C++ che andrò ad introdurre ora .

Quando una funzione viene richiamata dall'interno di un altra, il processore va ad eseguire una serie di operazioni sui registri e sullo stack per sapere a quale indirizzo del codice deve ritornare dopo l'esecuzione della funzione stessa.
Tale insieme di operazioni ovviamente necessita di un determinato tempo di esecuzione, quindi sarebbe meglio evitare no ? :P
In questo caso possiamo usare la direttiva "***inline***" del C nel seguente modo :

```
inline int somma( int a, int b )
{
   return a + b;
}

void funzione()
{
   int i;

   for( i = 0; i < 100000; i++ ){
       somma( i, i + 200 );
   }
}
```

Usando questa direttiva, diremo al compilatore di rimpiazzare ogni chiamata alla funzione "somma" con il suo codice, quindi come se non venisse richiamata la funzione, il che non costringe il processore ad eseguire quelle elaborazioni sullo stack che dicevo prima .
La stessa cosa si può fare usando la direttiva di precompilazione "define" per creare una macro :

```
#define somma(a,b) a + b

void funzione()
{
   int i;

   for( i = 0; i < 100000; i++ ){
       somma( i, i + 200 );
   }
}
```

Ovviamente queste sono solo le basi dell'ottimizzazione, generalmente parlando le cose fondamentali da ricordarsi sono strutturare bene i propri algoritmi, non eseguire operazioni superflue e usare le macro e inline dove è possibile .

Spero questo paper possa essere utile a qualcuno per capire come si scrive codice pulito e veloce, alla prossima, bye ;) .

evilsocket