Ingegneria degli Algoritmi
Corsi di Laurea in Ingegneria Informatica e Automatica
e Ingegneria dei Sistemi Informatici
Sapienza Università di Roma
A.A. 2012-2013

Gara di programmazione

Esercitatore: **Daniele Cono D'Elia**

Password cracking

In questo homework si propone di migliorare un'implementazione di un algoritmo per il cracking di password codificate attraverso `MD5`. Il codice allegato si presta a varie possibili ottimizzazioni attraverso una o più metodologie viste a lezione.

`MD5` è un algoritmo di hashing crittografico che codifica una stringa in input di lunghezza arbitraria in una di 128 bit che ne rappresenta il *digest* (detto anche *firma* o *checksum*). Proposto nel 1991 da Ronald Rivest e standardizzato attraverso la RFC 1321, questo algoritmo è caratterizzato da uno schema di codifica molto veloce e, sebbene importanti vulnerabilità identificate nel corso degli anni ne sconsiglino l'impiego, la sua diffusione risulta ad oggi ancora molto estesa.

Sorgente

Il codice sorgente allegato è strutturato in una serie di moduli C contenenti le funzioni essenziali per poter costruire un password cracker `MD5`.

Nel modulo `main.c` viene letto un file di input contenente per ciascuna riga il digest di una password da decifrare; essi andrannno a formare un array ordinato da dare in input al metodo `analyzeLength` - contenuto in `enum.c` - che genererà tutte le possibili combinazioni di lunghezza crescente a partire da un alfabeto e per ciascuna combinazione verificherà se il digest è presente nell'array.

Per generare le permutazioni utilizziamo il metodo ausiliario `encode` - contenuto in `enum.c` - che prende in input un alfabeto di `charsetLength` caratteri e genera la permutazione `counter`-esima di lunghezza `stringLength` a partire da un valore di riferimento memorizzato in `base`.

```
inline void encode(uint8_t* alphabet, size_t charsetLength, size_t stringLength,
    uint64_t counter, uint8_t* base, uint8_t* buffer ) {
  long j=0, a=0, carry=0;
  for ( ; j < stringLength; ++j, counter/= charsetLength) {
    a = base[j] + carry + (counter % charsetLength); // aggiorna cella j-esima
    carry = a / charsetLength; // tiene conto del riporto per lo step successivo
    a -= carry * charsetLength; // ... ma anche per quello corrente!
    buffer[j] = alphabet[a]; // costruisce la stringa
  }
}
```

La permutazione viene memorizzata in `buffer` secondo uno schema Little-Endian. Ad esempio, prendendo come set di caratteri `0123456789abcdef` ed invocando il metodo `encode` con valori per

`count` compresi tra 0 e 20, il metodo `provaGeneratore` riportato in basso genererà i primi 21 interi non negativi in formato esadecimale, con la cifra più significativa memorizzata in `buffer[1]`.

```
void provaGeneratore() {
  #define LENGTH 2
  uint8_t* base = calloc(LENGTH*sizeof(uint8_t), 1);
  uint8_t* buffer = malloc(LENGTH*sizeof(uint8_t));
  uint8_t* alphabet = (uint8_t*)"0123456789abcdef";
  long i, j;
  for (i=0; i<=20; ++i) {
    encode(alphabet, 16, LENGTH, i, base, buffer);
    for (j=LENGTH-1; j>=0; --j)
      printf("%c", buffer[j]);
    printf(" ");
  }
  printf("\n");
  free(base); free(buffer);
  #undef LENGTH
}
```

Per ciascuna combinazione prodotta, `analyzeLength` utilizza la funzione `callback` data in input per calcolarne il digest e procede quindi al confronto con gli hash delle password non ancora decifrate. Come funzione di callback utilizziamo il metodo `void md5(uint8_t *initial_msg, size_t initial_len, uint8_t *digest)` - presente in `md5.c` - che prende in input un array `initial_msg` formato da `initial_len` caratteri e scrive il digest corrispondente nel buffer `digest` che deve poter contenere 16 elementi di tipo `uint8_t`[1].

L'operazione di confronto viene effettuata attraverso il metodo `size_t findAndRemove(char** strings, char* s, size_t numLines)` - presente in `sort.c` - che effettua una ricerca binaria sulle prime `numLines` stringhe, assumendo che queste siano ordinate e che non contengano duplicati. Se la stringa non è presente il metodo restituisce `numLines`, altrimenti scambia l'elemento trovato con quello in posizione `numLines-1` (l'ultima stringa "valida" nell'array) e restituisce `numLines-1`.

Obiettivo

L'implementazione fornita del nostro `MD5` password cracker si presta a diverse ottimizzazioni che sfruttino le peculiarità di un calcolatore moderno.

Per semplicità assumiamo che le password possano essere formate da lettere minuscole e da numeri. Il codice allegato è in grado di analizzare poco più di 400000 combinazioni al secondo su un Intel Core i7-3632QM@2.20GHz, compilato con `-O3` con `gcc 4.7.2` su Ubuntu 12.10 a 32 bit. Attraverso una semplice stima su questi parametri[2] si evince che possono essere necessari fino a 90 minuti per testare tutte le combinazioni di lunghezza 6, ma fino a 82 giorni per combinazioni di lunghezza 8!

Per avere informazioni statistiche è possibile usare la versione del tool `md5crack_verbose` compilata con `-DVERBOSE`:

```
$ echo -n "lea89" | md5sum >> password_list.txt
$ ./md5crack_verbose password_list.txt
```

[1] Il tipo `uint8_t` è compatibile con un `unsigned char`.

[2] `http://lastbit.com/pswcalc.asp`

```
[info] analyzing passwords of length 1...
[info] analyzing passwords of length 2...
[info] analyzing passwords of length 3...
[info] analyzing passwords of length 4...
[info] analyzing passwords of length 5...
[info] analyzed passwords per second: 410164.20
[info] analyzed passwords per second: 410866.60
[info] analyzed passwords per second: 409715.51
[info] analyzed passwords per second: 405489.35
[info] analyzed passwords per second: 410015.94
[info] analyzed passwords per second: 412050.07
[info] analyzed passwords per second: 407849.23
cf3467b0b8b3c6c94e6160d5c8b19782 lea89
All passwords have been cracked!
```

Per la creazione di input di prova, il flag `-n` per il comando `echo` previene l'aggiunta alla stringa del carattere di nuova linea "`\n`", che cambierebbe il digest associato alla stringa. Su sistemi Mac OS X, invece, è sufficiente eseguire il comando "`md5 -q -s lea89 >> password_list`", dove "lea89" è ad esempio la password da "craccare". Ricordiamo che per il corretto funzionamento del tool non devono esserci linee duplicate all'interno del file.

Test

Il `Makefile` in dotazione permette di compilare il programma di password cracking con `make` ed eseguire un test con `make test` su piattaforme Linux e `make test-osx` su Mac OS X.

Valutazione

Si faccia riferimento al sito web del corso `http://www.dis.uniroma1.it/~demetres/didattica/ae` per gli aspetti legati alle modalità della gara, al voto di esame e ai tempi di consegna.

Approfondimenti e link utili

[1] RFC1321 con specifica dettagliata di `MD5`
`http://tools.ietf.org/html/rfc1321`

[2] Introduzione alle funzioni hash crittografiche e ai message digest
`http://www.unixwiz.net/techtips/iguide-crypto-hashes.html`

[3] Esempio di collisione sul checksum `MD5` di due eseguibili
`http://www.mathstat.dal.ca/~selinger/md5collision/`

[4] Aggiunta del *sale* per rendere più sicuro l'hashing delle password
`http://crackstation.net/hashing-security.htm`

[5] Tool gratuito avanzato per password cracking GPU-based di numerosi algoritmi
`https://hashcat.net/oclhashcat-plus/`