"Do to others what you want them to do to you."

64 Cracking ZIP file's Password

We all use ZIP files (compressed files) for saving disk space. PKZIP is one of the famous ZIP utility. PKZIP provides us security mechanism to save the contents of ZIP file being viewed by others. For this security mechanism one has to use passwords. These passwords can be cracked with dictionary attacks. The encryption algorithm uses case sensitive passwords.

The very famous Windows based WinZip also uses the same compression algorithm used by PKZIP. So there is no difference between the ZIP file created with WinZip and PKZIP.

64.1 Cracking ZIP passwords

In order to crack the ZIP file's passwords, you need to know the file format of ZIP. So I suggest you to have a look at the ZIP file format found on file formats section.

64.2 CrackIt

64.2.1 Logic

The following code is very popular among crackers. Let's call it as *CrackIt* utility. CrackIt uses dictionary attack technique. So you need to provide a *Words list* file that is preloaded with all the passwords you suspect. For example, if you suspect that the password would be any one of the words "KING", "QUEEN", "JACK", you have to load the *Words list* file as:

```
KING
QUEEN
JACK
```

The CrackIt would first take the "KING" and it would check whether it is the right password or not. If not, it would check "QUEEN" and if it is the right password, it would print it. The validation of password is done with dictionary attack.

The encryption algorithm uses case sensitive passwords. So you have to load the *Words list* file with enough words list. A clever idea is to use brute force for preparing words list that are to be used in *Words list* file.

CrackIt has few drawbacks:

1. Success of the cracking depends upon the *Words list* file
2. Dictionary attack won't be faster if you use large *Word list*

3. Because of the encryption mechanism used in PKZIP, it requires at least 3 enciphered files be present in a given ZIP file

64.2.2 Code

Following is the code for CrackIt. To check it run it as:

```
C:\>CRACKIT FOO.ZIP WORDS.LST
```

```
#include <stdio.h>

#define  ZIP      (0x04034b50)      /* signature */

typedef int BOOLEAN;

#define TRUE      (1)
#define FALSE     (0)

unsigned long Crc32_Tbl[] =
     {
      0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL,
      0x076dc419L, 0x706af48fL, 0xe963a535L, 0x9e6495a3L,
      0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L,
      0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
      0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
      0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L,
      0x136c9856L, 0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL,
      0x14015c4fL, 0x63066cd9L, 0xfa0f3d63L, 0x8d080df5L,
      0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L, 0xa2677172L,
      0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
      0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L,
      0x32d86ce3L, 0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L,
      0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L,
      0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L, 0xb8bda50fL,
      0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
      0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL,
      0x76dc4190L, 0x01db7106L, 0x98d220bcL, 0xefd5102aL,
      0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L, 0xe8b8d433L,
      0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L,
      0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
      0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL,
      0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L,
      0x65b0d9c6L, 0x12b7e950L, 0x8bbeb8eaL, 0xfcb9887cL,
      0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L, 0xfbd44c65L,
      0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
      0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL,
      0x4369e96aL, 0x346ed9fcL, 0xad678846L, 0xda60b8d0L,
```

```
        0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L,
        0x5005713cL, 0x270241aaL, 0xbe0b1010L, 0xc90c2086L,
        0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
        0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L,
        0x59b33d17L, 0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL,
        0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL, 0x74b1d29aL,
        0xead54739L, 0x9dd277afL, 0x04db2615L, 0x73dc1683L,
        0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
        0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L,
        0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL,
        0xf762575dL, 0x806567cbL, 0x196c3671L, 0x6e6b06e7L,
        0xfed41b76L, 0x89d32be0L, 0x10da7a5aL, 0x67dd4accL,
        0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
        0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L,
        0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL,
        0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0x41047a60L,
        0xdf60efc3L, 0xa867df55L, 0x316e8eefL, 0x4669be79L,
        0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
        0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL,
        0xc5ba3bbeL, 0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L,
        0xc2d7ffa7L, 0xb5d0cf31L, 0x2cd99e8bL, 0x5bdeae1dL,
        0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL,
        0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
        0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L,
        0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L,
        0x86d3d2d4L, 0xf1d4e242L, 0x68ddb3f8L, 0x1fda836eL,
        0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L, 0x18b74777L,
        0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
        0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L,
        0xa00ae278L, 0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L,
        0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e6e77dbL,
        0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L, 0x37d83bf0L,
        0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
        0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L,
        0xbad03605L, 0xcdd70693L, 0x54de5729L, 0x23d967bfL,
        0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L, 0x2a6f2b94L,
        0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
      };

#define CRC32( x, y ) (Crc32_Tbl[((int)(x) ^ (y)) & 0xff] ^ ((x) >> 8))

int main( int  argc, char *argv[] )
{
   BOOLEAN  tried_all, found;
   int  byte;
   int  byte_num;
   long  compressed_size;
```

```
int  extra_field_length;
char  file_name[1024];
int  file_name_length;
int  file_num;
int  flags;
unsigned char  header[3][12];
unsigned long  key[3];
int  num_enciphered;
char  password[255];
char  *password_ptr;
long  signature;
unsigned char  target[3];
int  tem;

FILE  *wordlist_fp, *zip_fp;

if ( argc < 3 )
  {
    printf( "Syntax: CRACKIT <zipfile> <wordslistfile> \a\n " );
    exit(1);
  }
/* Check for file errors....*/
if ( (zip_fp=fopen(argv[1], "rb")) == NULL )
   {
   printf( "Error:  Couldn't open %s \a\n", argv[1] );
   exit(1);
   }
if ( (wordlist_fp=fopen(argv[2], "r") ) == NULL )
   {
   printf( "Error:  Couldn't open %s \a\n", argv[2] );
   exit(1);
   }
/* <- checked ok */

/* Read the necessary informations from ZIP file... */
num_enciphered = 0;
while ( (num_enciphered < 3)
   /* Read 4 bytes from ZIP file... */
     && fread( &signature, 4, 1, zip_fp )
     && (signature == ZIP) )
  {
    fseek( zip_fp, 2L, SEEK_CUR );
    fread( &flags, 2, 1, zip_fp );
    if ( flags & (1<<0) )   /* bit0 set? */
    {
      fseek( zip_fp, 9L, SEEK_CUR );
      fread( &(target[num_enciphered]), 1, 1, zip_fp );
```

```
        fread( &compressed_size, 4, 1, zip_fp );
        fseek( zip_fp, 4L, SEEK_CUR );
        fread( &file_name_length, 2, 1, zip_fp );
        fread( &extra_field_length, 2, 1, zip_fp );
        fread( &file_name[0], 1,
        file_name_length, zip_fp );
        file_name[file_name_length] = '\0';
        fseek( zip_fp, (long)extra_field_length, SEEK_CUR );
        fread( &(header[num_enciphered++][0]), 1, 12, zip_fp );
        compressed_size -= 12L;
        printf( "%s is enciphered \n", &file_name[0] );
      }
     else
       {
         fseek( zip_fp, 10L, SEEK_CUR );
         fread( &compressed_size, 4, 1, zip_fp );
         fseek( zip_fp, 4L, SEEK_CUR );
         fread( &file_name_length, 2, 1, zip_fp );
         fread( &extra_field_length, 2, 1, zip_fp );
         compressed_size += file_name_length+extra_field_length;
       }
      fseek( zip_fp, compressed_size, SEEK_CUR );
   }
fclose(zip_fp);



if (num_enciphered == 0)
    printf( "Nothing is enciphered in %s \n", argv[1] );
   else if (num_enciphered < 3)
    {
      printf( "Less than 3 files are enciphered in %s \a\n",
                                          argv[1] );
      printf( "CRACKIT requires atleast 3 enciphered files \n" );
    }
     else /* Crack using wordlist....*/
      {
          found = FALSE;
          byte_num = 0;
          while (fgets(&password[0],255,wordlist_fp) != NULL)
           {
            password[strlen(&password[0])-1] = '\0';
            tried_all = TRUE;
            file_num = 0;
            while (tried_all && (file_num<num_enciphered))
             {
               key[0] = 305419896L;
```

```
                key[1] = 591751049L;
                key[2] = 878082192L;
                password_ptr = &password[0];
                while (*password_ptr != '\0')
                 {
                 byte = *(password_ptr++);
                 key[0] = CRC32( key[0], byte );
                 key[1] += key[0] & 0xff;
                 key[1] = key[1]*134775813L + 1;
                 key[2] = CRC32( key[2], key[1] >> 24);
                 }
                for ( byte_num=0; byte_num < 12; ++byte_num )
                 {
                 tem = key[2] | 2;
                 byte = header[file_num][byte_num]
                          ^(((tem*(tem^1)) >> 8) & 0xff);
                 key[0] = CRC32( key[0], byte );
                 key[1] += key[0] & 0xff;
                 key[1] = key[1]*134775813L + 1;
                 key[2] = CRC32( key[2], key[1] >> 24 );
                 }
                if ( byte == target[file_num] )
                    ++file_num;
                 else
                    tried_all = FALSE;
              }
             if ( tried_all )
               {
                  if (!found)
                   {
                      found = TRUE;
                      printf( "Passwords migh be: \n" );
                   }
                  printf( "\t %s \n", &password[0] );
               }
           }

          if (!found)
                printf( "%s don't hold the right Password \a\n",
                                                 argv[2] );
          fclose(wordlist_fp);
       }

   return(0);
} /*--main( )------*/
```