# Flare-On 3: Challenge 4 Solution

**Challenge Author: James T. Bennett**

Challenge 4 is a DLL with 50 functions exported by ordinal. Besides the exports and DLLMain, the DLL only contains two other functions; both of which appear to be cryptographic in nature. Using the signsrch plugin for IDA Pro identifies them as being related to the CAST-128 cipher. These two functions are called from export ordinal 51 and indirectly from export ordinal 50. These two uses of CAST-128 are for different ciphertexts and use different keys, all of which are stored in global variables. There is a cross-reference for the export ordinal 51 key from every other export except ordinal 50. The key in export ordinal 50 is only referenced directly from export ordinal 50 itself. The 48 other exports that reference the first key are all modifying the key at a specified global index and decrementing the index by 1. At this time, it seems that the goal should be to call these 48 exports in some order and then call export ordinal 51 to decrypt the key, or something that will help us find the key. Since these exports have no cross-references, it becomes clear that the order in which they are to be called must be determined in another manner. Examining these exports more closely, they appear to be generated automatically and follow a distinct pattern. They each have two local integer values that are assigned and then XORed against each other for the return value. These integer values are always quite small, and in fact appear to stick within the range of the ordinal values present in this DLL. If the return values are in fact hints as to the next ordinal to call, the matter becomes simply determining which export to call first. We can determine this by writing a small IDApython script, as shown in Figure 1, to build a sorted list of return values. The ordinal not in the list is our point of departure: 48.

```
import re
ret = []
rptn = r"^flareon2016challenge_([\d]+)$"
for f in Functions():
    m = re.match(rptn, GetFunctionName(f))
    if m != None and m.group(1) != "51":
        nextordinal = GetOperandValue(f+0x07, 1) ^ GetOperandValue(f+0x0E, 1)
        ret.append(nextordinal)
ret.sort()
for i in range(1, 49):
    if ret[i-1] != i:
        print i
        break
```

Figure 1: IDApython script to find first export ordinal

Now we have everything we need to write a small program to call the exports in their proper order and dump the decrypted data to a file for further inspection. Such a program is shown in Figure 2.

```
#include <Windows.h>
#include <stdio.h>
#include <stdint.h>

typedef uint32_t (*EXP)(
      VOID
);

int main(int argc, char **argv)
{
      HMODULE hLib;
      HANDLE hFile;
      EXP exp;
      uint32_t plaintxt;
      uint32_t nextord;
      hLib = LoadLibraryA((LPCSTR)argv[1]);
       nextord = 48; //our starting point
      //decrypt key
       while(nextord != 51)
      {
            exp = (EXP)GetProcAddress(hLib, MAKEINTRESOURCE(ret));
            nextord = exp();
      }

      exp = (EXP)GetProcAddress(hLib, MAKEINTRESOURCE(51));
      //decrypt plaintxt
      exp();
      //dump plaintext to file
      plaintxt = *(uint32_t*)((uint32_t)exp+0x2e);
      hFile    =    CreateFile("out",    GENERIC_WRITE,    0,    0,    CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0);
      WriteFile(hFile, (LPCVOID)plaintxt, 0x1A10, 0, 0);
      CloseHandle(hFile);
      return 0;
}
```

Figure 2: Program to decrypt and dump data

The decrypted data is a simple executable that makes a series of calls to the Beep API. This is interesting because export ordinal 50 also makes a call to Beep and export ordinal 51 prints the hint `play me a song, have me play along`. It seems like we have found our song to play! Now all that needs to be done is to write another small program to call export ordinal 50 to have it play the song we discovered in the embedded executable. The export will print what we hope is the final key for us. Figure 3 shows such a program.

```
#include <Windows.h>
#include <stdio.h>
#include <stdint.h>

typedef VOID (*BEEPBEEP)(
	DWORD dwFreq,
	DWORD dwDuration
);

int main(int argc, char **argv)
{
	HMODULE hLib;
	BEEPBEEP beep;
	hLib = LoadLibraryA((LPCSTR)argv[1]);
	beep = (BEEPBEEP)GetProcAddress(hLib, MAKEINTRESOURCE(50));
	beep(440,500);
	beep(440,500);
	beep(440,500);
	beep(349,350);
	beep(523,150);
	beep(440,500);
	beep(349,350);
	beep(523,150);
	beep(440,1000);
	beep(659,500);
	beep(659,500);
	beep(659,500);
	beep(698,350);
	beep(523,150);
	beep(415,500);
	beep(349,350);
	beep(523,150);
	beep(440,1000);

	return 0;
}
```

Figure 3: Program to play the song and decrypt the final key

And that it does! Our key is `f0ll0w_t3h_3xp0rts@flare-on.com`