Handling Binary Formats

Description

This lab aims to practice handling binary formatted data structures and files. Please follow the steps below to complete this lab.

1. Please download a challenge file from this URL:
   https://inp.zoolab.org/binflag/challenge?id=<your-student-id>
   You can use tools like wget or curl to download your own challenge file. Remember to use your student id to download the file.
2. The downloaded challenge file is a customized file with the following file formats. It is composed of a file header, a number of data blocks, and the flag information. You can reconstruct a ***dictionary file*** D using the blocks in the challenge file. You can then retrieve the flag hidden in D using the flag information structure. Please read the formats carefully. A sample figure to illustrate the layout of the challenge file is as follows.
3. File header: The file header is defined using the structure:

```
typedef struct {
    uint64_t magic;      /* 'BINFLAG\x00' */
    uint32_t datasize;   /* in big-endian */
    uint16_t n_blocks;   /* in big-endian */
    uint16_t zeros;
} __attribute((packed)) binflag_header_t;
```

   The magic is a fixed string of value BINFLAG\x00. The datasize is the total size of the dictionary file D. The n_blocks is the total number of blocks in the challenge file. Note that some blocks might be bad blocks, i.e., having an incorrect checksum. You should not consider bad blocks when reconstructing D.
4. Based on the n_blocks defined in the file header, there are n_blocks data blocks placed right after the file header. The data structure for each data block is as follows.

```
typedef struct {
    uint32_t offset;      /* in big-endian */
    uint16_t cksum;       /* XOR'ed results of each 2-byte unit in payload */
    uint16_t length;      /* ranges from 1KB - 3KB, in big-endian */
    uint8_t  payload[0];
} __attribute((packed)) block_t;
```

   The offset is the offset of the payload in D. The length is the length of the payload. By using the offset and the length of each data block, you have to copy the payload of a data block to the corresponding offset in D. Note that

you have to check the validity of a data block by using the cksum field. In case a data block does not pass the check, you should not use the bad data block.

5. The flag information is placed right after all data blocks. The data structure for the flag information is as follows.

   typedef struct {
   uint16_t length; /* length of the offset array, in big-endian */
   uint32_t offset[0]; /* offset of the flags, in big-endian */
   } __attribute((packed)) flag_t;

   With the flag information and content of the resulting dictionary file D, you should be able to retrieve the flag by using the offset values in the offset array. Each offset in the flag information points to ***two successive flag characters*** in D. By concatenating the flag characters pointed by the offsets in order, you should get the complete flag.
6. Convert your flag into a hexadecimal string and send it to the flag verification service at:

   https://inp.zoolab.org/binflag/verify?v=<your-flag-in-hexadecimal>

   Suppose the FLAG is FLAG|aaa|bbb|ccc. You should convert the flag to 464c41477c6161617c6262627c636363 and submit it to our flag verification service. Note that the FLAG|aaa|bbb|ccc is not a valid flag and cannot pass the flag verification service's check. Here is another demo flag that can pass the verification:464c41477c87b30fbafbbdbcf6188f00e4291c8aa1
7. The regular flag extracted from a downloaded challenge server is in the format.

   FLAG|id|datetime|random-bytes

   You may play with our demo challenge file from here (demo2.bin).
8. You have to implement the challenge file decoder program in C or C++.
9. A minimized demo file can be downloaded from here (demo1.bin). It has only 286 bytes, and the hex dump of this file is as follows.

   OFF <file header>
   000 42 49 4e 46 4c 41 47 00 00 00 00 80 00 06 00 00 |BINFLAG.....|
   <blocks (6)>
   010 00 00 00 00 91 56 00 20 75 4c f9 5b 17 aa 1e e8 |.....V. uL.[....|
   020 4d 8f 3c d0 0a be bd 9c 23 0b 3e 7b 24 3c d1 8b |M.<.....#.>{$<..|
   030 f6 50 14 42 ca eb 12 40 00 00 00 18 09 54 00 10 |.P.B...@.....T..|
   040 d3 bd 5f d7 6a ee 8a fd 39 ec 81 99 66 1a da 82 |.._.j...9...f...|
   050 00 00 00 40 3b bb 00 30 46 4c 27 19 38 77 08 ad |...@;..0FL'.8w..|
   060 7a 80 36 c0 fc 9c 70 7f ca bf 07 60 e6 da a9 a2 |z.6...p....`....|
   070 7b 28 fb 6f fb 4b 9c 0b f0 a4 41 47 20 0b a0 a1 |{(.o.K....AG ...|
   080 70 a1 6c 60 03 0b e1 b4 00 00 00 3e cd bb 00 20 |p.l`.......>... |

090 c2 dc b3 1a 43 e0 fe a4 94 88 67 27 65 ca e0 d6 |....C.....g'e...|
0a0 0d 1e bc 79 aa 0a f4 41 e6 94 c8 25 f2 ec 6d c9 |...y...A...%..m.|
0b0 00 00 00 70 19 42 00 10 30 c2 1c ef 5e 04 05 63 |...p.B..0...^..c|
0c0 1d 44 94 a1 63 26 84 cb 00 00 00 20 5e 4e 00 20 |.D..c&..... ^N. |
0d0 4a 25 49 c9 ac fb e5 90 5b 3a 6b f2 7a 01 02 9c |J%I.....[:k.z...|
0e0 05 3f be 16 77 05 0f 30 cf dd 7c 78 6f 4d 43 68 |.?..w..0..|xoMCh|
<flag information>
0f0 00 0b 00 00 00 40 00 00 00 62 00 00 00 3a 00 00 |.....@...b...:..|
100 00 56 00 00 00 1a 00 00 00 0a 00 00 00 5c 00 00 |.V.....\..|
110 00 5e 00 00 00 0e 00 00 00 08 00 00 00 60 |.^.....`|

The challenge file contains 6 blocks, and the dictionary file reconstructed from the challenge file is 128 bytes. The flag information indicates that there are 11 offsets in the sequence [64, 98, 58, 86, 26, 10, 92, 94, 14, 8, 96]. The flag hidden in the challenge file is 464c41477c78a9a214423cd0fb4b9c0bbd9c4d8ff0a4 (or a byte string :

FLAG|x\xa9\xa2\x14B<\xd0\xfbK\x9c\x0b\xbd\x9cM\x8f\xf0\xa4).

Demonstration

1. [20%] You can decode the flag from the demo2.bin file and pass the verification service's check.
2. [70%] You can decode the flag downloaded from the challenge server and pass the verification service's check.
3. [10%] You can implement a single command to a) download a challenge file from the server using your id, b) decode the flag, and c) verify the flag. You can implement a shell script to perform the automation, but step (b) should be performed by the program you implemented in (2).