Inspecting Executables

Programs are simply binary files whose contents can be interpreted as a set of machine instructions. Thus we can write programs that inspect and modify them.

For instance the Unix program `objdump`[1] displays sections of executables in human readable form. Called on a compiled "hello world" program, `objdump` produces:

```
$ objdump -s -j .rodata helloworld

hello:     file format elf64-x86-64

Contents of section .rodata:
 4005d0 01000200 68656c6c 6f20776f 726c6400  ....hello world.
```

The flags to `objdump` enable us to look at the read-only section of the executable where string literals (among other things) are stored. The output above is formatted as follows:

- The first column contains memory addresses. In the output above, the `.rodata` section starts at address `4005d0`.
- The next 4 columns contain the hexadecimal representations of the memory contents at those addresses. Recall that two hexadecimal digits together represent one byte, so we can see that the `.rodata` section of this program is 16 bytes long (for a larger program, `objdump` will split up the output across multiple 16-byte rows).
- The last column also displays the memory contents, but in ASCII representation—note that some of the bytes are printable as valid ASCII characters, whereas other bytes are not (and are thus represented by a `.` instead), since the `.rodata` section can contain other data besides string literals.

However `objdump -s -j` itself is insufficient for locating `.rodata` inside the executable file, as the addresses it lists are **LMAs** (Load Memory Addresses[2]), rather than **file offsets**. This file offset we need (as seen under the `File off` column) is found by doing

```
$objdump -h -j .rodata helloworld

hello:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
 15 .rodata       00000010  00000000004005d0  00000000004005d0  000005d0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

For this example, the output above tells us that the `.rodata` section can be found in the executable file starting from byte `05d0`.

[1] Objdump comes included in the GNU Binary Utilities. You may be interested in some of the other included tools, as well: `https://www.gnu.org/software/binutils/`

[2] LMAs specify the memory addresses of where each section of the executable file should be loaded by the operating system.

Question 1. Write a program `literals.c` which takes three command-line arguments:

1. address in hexadecimal of the first byte in the rodata section,
2. size (number of bytes) in decimal of the rodata section, and
3. name of an executable file

that prints all of the string literals stored in the rodata section, one per line (notice in the sample output below that the first two lines appear to be garbage, since there is some non-string data that is stored in the `rodata` section before the "hello world" literal, as can be seen in the `objdump` output above):

```
$ ./literals 0x000005d0 16 helloworld
^A
^B
hello world
```

- Use base-16 `strtol` to parse hexadecimal numbers that lead with `0x`.
- Read in the entire rodata section first then think about how to print out individual strings on new lines. Remember that there could be many null-terminated strings in the rodata section.

Question 2. For `literals` to be useful we need a program to determine the location and size of the rodata section of a given executable. It should not be surprising that this data is encoded in the executable itself. Here is the relevant information **for basic programs compiled using gcc on the lab machines**:

- The four-bytes starting at address `0x28` contains the starting address for all the section headers.
- Each section header is 64-bytes long and the section header for the rodata section is the 16th header. Use this to compute the staring address of the rodata section header.
- Each section header has integers stored in addresses 24-27 and 32-35 bytes after its starting address. These two integers store the address and size of the section data, respectively.

Create a program `rodata.c` which takes the filename of an executable and prints the address of the rodata section (in hex) and the size of the section (in decimal).

```
$ ./rodata hello
0x000005d0 16
```

Question 3. Modify `literal` to accept output from `rodata` through a pipe.

```
$ ./rodata hello | ./literals hello
```