Lab 6: Decoding Binary Code

Date: Oct 12, 2021

This document first describes the aims of this lab. It then describes the exercises which need to be performed.

0.1 Aims

The aim of this lab is to familiarize you with using tools to decode binaries. After completing this lab, you should have some familiarity with the following topics:

- Using gdb to disassemble programs and examine memory and registers.
- Using the `objdump` utility to disassemble programs.

0.2 Exercises

0.2.1 Starting up

Follow the *provided directions* for starting up this lab in a new git `lab6` branch and a new `submit/lab6` directory. You should have copied over the contents of `~/cs220/labs/lab6/exercises` over to your directory.

In this lab you will try to feed a cookie monster. First, generate a cookie for the cookie monster:

```
$ ./gen-cookie
$ cat .cookie
```

This should show you the generated cookie; the cookie value is based on your user-id.

In this lab, some code will wrap the cookie and then you will need to provide a key to unwrap the cookie to feed the cookie monster. For example:

```
$ cd exercises/full-example
$ make -f ../Makefile
gcc -g -Og -Wall -std=c18    -c \
  -o feed-cookie-monster.o feed-cookie-monster.c
gcc feed-cookie-monster.o cookie-wrapper.obj \
  -o feed-cookie-monster

# an unsuccessful attempt
```

```
$ ./feed-cookie-monster
Me want cookie!
enter cookie unwrap key in hex: 9a03
 AARGH YUCK!!

# a successful attempt
$ ./feed-cookie-monster
Me want cookie!
enter cookie unwrap key in hex: 7a00
 CHOMP!!! CHOMP!!!
```

The code for unwrapping the cookie is only available as binary in a `.obj` file as a function **unwrapCookie**`()` with specification **int unwrapCookie**(**int** `wrapped`, **int** `key`). You will need to use tools to disassemble the code.

The main program you will be running is in feed-cookie-monster.c and the interface to the binary file is in cookie-wrapper.h. Study these two files right now. You will need to figure out what the `unwrapCookie()` function does so that you can figure out what `key` you should give it so that it returns a cookie to feed the cookie monster.

0.2.2 Exercise 1: Guided Decoding of Binary

Change to the full-example directory and build the program.

```
$ make -f ../Makefile
gcc -g -Og -Wall -std=c18   -c \
  -o feed-cookie-monster.o feed-cookie-monster.c
gcc feed-cookie-monster.o cookie-wrapper.obj \
  -o feed-cookie-monster
$
```

Important Note: The following gdb traces were generated assuming that the cookie value was `0x7a01`. The values in the gdb traces depend on this value; hence the traces you will see will most likely be different from those given below.

Start the debugger on the `feed-cookie-monster` executable:

```
$ gdb feed-cookie-monster
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
...
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from feed-cookie-monster...done.
```

Put a breakpoint on `unwrapCookie()`, start the program, and provide some value for the `key`:

```
(gdb) b unwrapCookie       #put a breakpoint
Breakpoint 1 at 0x14eb
(gdb) r                    #start program
Starting program: .../feed-cookie-monster
Me want cookie!
enter cookie unwrap key in hex: 33   #provide key of 0x33

Breakpoint 1, 0x00005555555554eb in unwrapCookie ()
```

When the breakpoint is entered you are positioned at the start of the `unwrapCookie()` function. Disassemble it:

```
(gdb) disass
Dump of assembler code for function unwrapCookie:
=> 0x00005555555554eb <+0>:     endbr64
   0x00005555555554ef <+4>:     lea    (%rdi,%rdi,2),%eax
   0x00005555555554f2 <+7>:     and    $0xf,%eax
   0x00005555555554f5 <+10>:    lea    0x2b64(%rip),%rdx \
                               # 0x555555558060 <vals>
   0x00005555555554fc <+17>:    add    (%rdx,%rax,4),%esi
   0x00005555555554ff <+20>:    mov    %esi,%eax
   0x0000555555555501 <+22>:    retq
End of assembler dump.
```

Verify the above disassembly using objdump. In another terminal:

```
$ objdump -d feed-cookie-monster
```

You should see the disassembled code for `unwrapCookie()` along with the code for other functions. It should essentially be the same as what you got above.

Return to the terminal running gdb. Note that the gdb disassembler is helpfully telling us that the address `0x2b64(%rip)` corresponds to address `0x55555555-8060` which corresponds to some symbol `vals`. Let's verify this (note that `rip` will point to the address of the next instruction which is `0x00005555555554fc`):

```
(gdb) p /x 0x00005555555554fc + 0x2b64
$2 = 0x555555558060
```

which checks out.

We know that `unwrapCookie()` takes two arguments; hence `rdi` must contain the `wrapped` value and `rsi` must contain the `key` provided above. Let's check:

```
(gdb) i reg $rdi
rdi            0x7                7
(gdb) i reg $rsi
rsi            0x33               51
```

So it looks like `wrapped` is `0x7` and `key` is `0x33`. The latter matches the provided key. So we need to figure out a new value for `key` so that the function returns our cookie value `0x7a01`.

Annotating the above code:

```
# rdi = wrapped; rsi = key
unwrapCookie:
lea    (%rdi,%rdi,2),%eax  # eax = 3*rdi = 3*wrapped
and    $0xf,%eax           # eax &= 0xf
lea    0x2b64(%rip),%rdx \ # point rdx to vals
        # 0x555555558060 <vals>    # at this address
add    (%rdx,%rax,4),%esi  # esi += rdx[(int)eax]
mov    %esi,%eax           # eax = esi
retq
```

So it looks like `unwrapCookie()` is returning something like `key + vals[(3*¬ wrapped)&0xf]`. The incoming `wrapped` is `0x7`. Hence `(3*wrapped) & 0xf` is `0x5`. We need to figure out what the value of `vals[5]` is. Note that since the index for `vals[]` is guaranteed to be less than 16 (because of the `&0xf`), let's look at that location as 16 `int`'s.

```
(gdb) p ((int *)0x555555558060)[0]@16
$1 = {12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

So `vals[5]` is `1`. Hence the function returns `key + 1`. Since we want it to return the value of our cookie, namely `0x7a01`, we should provide `key` as `0x7a00`. Let's try that:

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: .../feed-cookie-monster
Me want cookie!
enter cookie unwrap key in hex: 7a00

Breakpoint 1, 0x00005555555554eb in unwrapCookie ()
```

```
(gdb) c
Continuing.
 CHOMP!!! CHOMP!!!
[Inferior 1 (process 1281943) exited with code 01]
(gdb) q
$
```

That worked!!

0.2.3 Exercise 2: A Simple Decode

Feed the cookie monster using the files provided in simple. Proceed as in the previous exercise.

0.2.4 Exercise 3: Decode with Data

Feed the cookie monster using the files provided in another-one. Proceed as in the previous exercises.