RCE - 100pts

So checking the binary that we got at verbosity we can see that it sets a signal handler for all signals.

```
for ( sig = 0; sig <= 63; ++sig )
  signal(sig, generic_handler);
alarm(5u);
client_fd = v12;
doprocessing(v12);
```

So every time we managed to crash the server due to a buffer overflow or something else we got that:

```
if ( client_fd != -1 )
{
  puts("Closing");
  write(client_fd, "Nice try but no.\n", 0x11uLL);
  sleep(1u);
}
```

So I started looking at how the get command is parsed and I noticed this bug.

```
len_ = len;
v30 = *MK_FP(__FS__, 40LL);
src = request_buf;
needle = -724252512;
v9 = -32560;
v10 = 0;
dexor(&needle, 7LL);
v12 = memmem(request_buf, len_, &needle, 4uLL);
if ( v12 )
{
  *(_BYTE *)v12 = 0;
  memcpy(&dest, src, len_);
```

We copy in dest which is a fixed stack buffer, a user controlled value.

```
__int64 __fastcall server_get(int client_fd, void *request_buf, int len)
{
  int len_; // [sp+8h] [bp-1E8h]@1
  int fd; // [sp+1Ch] [bp-1D4h]@12
  int v6; // [sp+20h] [bp-1D0h]@15
  signed int v7; // [sp+24h] [bp-1CCh]@3
  int needle; // [sp+30h] [bp-1C0h]@1
  __int16 v9; // [sp+34h] [bp-1BCh]@1
  char v10; // [sp+36h] [bp-1BAh]@1
  void *src; // [sp+38h] [bp-1B8h]@1
  void *v12; // [sp+40h] [bp-1B0h]@1
  char *s; // [sp+48h] [bp-1A8h]@3
  char *v14; // [sp+50h] [bp-1A0h]@3
  void *v15; // [sp+58h] [bp-198h]@7
  char dest; // [sp+60h] [bp-190h]@3
```

Luckily for the server there is a stack cookie that prevents us from overflowing it easily. But as we saw before this is a fork() server.

```
v13 = fork();
if ( v13 < 0 )
{
  perror("ERROR on fork");
  exit(1);
}
if ( !v13 )
{
  close(v10);
  for ( sig = 0; sig <= 63; ++sig )
    signal(sig, (__sighandler_t)generic_handler);
  alarm(5u);
  client_fd = v12;
  doprocessing(v12);
  exit(0);
}
```

That means that the canary remains the same for every connection to the server. So guided by the output I wrote a canary brute forcer. Once the canary was taken care of the last part is to write our system(/bin/sh) rop. To do that we first have to leak a libc address. I did that by overwriting the return address of the server_get function with write(5, write_got, random_value_which_I_could_not_control).

While writing the rop I noticed that I did not have enough space to write both dup2(5,0) and dup2(5,1) so I just redirected the input. In the end I had a shell with no output.

I checked that I have a shell by creating a file in /tmp and accessing it from the webpage, which worked like a charm. To finish the task I used the first reverse shell that I found on google sent it to the server via my VPS and compiled it there. Then I used that to connect back to my VPS and get a normal shell on the machine and read the flag.

reveal - 100pts

In this task the flag was the md5 of the remote-machine ip.
Using the shell from the previous task I did:
$ ifconfig