solved 5 times

[j00ru] Antipasto (pwn 50)

- Was supposed to be an easy task.
  - Turned out: not so much.
- Linux, 32-bit, no NX, no PIE, no RELRO, `stdin`/`stdout` redirected.
- Simple echo logic in a loop:
  - Read uint32 (N) from input.
  - Read N bytes from input to static buffer.
  - Write the N bytes back.

[j00ru] Antipasto (pwn 50)

```
void ReadBytes(char *buffer, unsigned int bytes_no) {
  unsigned int bytes_read = 0;

  while (bytes_read != bytes_no) {
    ssize_t ret = read(STDIN_FILENO, &buffer[bytes_read], bytes_no - bytes_read);
    if (ret == 0) {
      exit(1);
    }

    bytes_read += ret;
  }
}
```

[j00ru] Antipasto (pwn 50)

- Typical overflow with controlled size, but there is nothing to overwrite after the statically located buffer - dead end.
- However, return value of read() is not fully checked.
  - May return -1 when an error occurs.
  - Example: when the syscall tries to write outside mapped memory.
  - When error code is returned, the internal position in the fd does not change; a subsequent read() will return the same data.
  - In our task, a return value of -1 results in shifting the pointer *backwards* (into the .got.plt section).

[j00ru] Antipasto (pwn 50)

```
bytes_no = address of .data.buffer - .got.plt.write

data = dd(.got.plt.write + 4) + shellcode
```

- `read()` will read the data to `&buffer[0]`, `&buffer[-1]`, `&buffer[-2]` … until it reaches the .got.plt.write function pointer.
- `bytes_read == bytes_no` is satisfied, the loop exits, overwritten `write` pointer is called.
- No PIE or NX, so you can just have shellcode directly in the static memory and point to it in the function pointer.

[j00ru] Antipasto (pwn 50)

- Spawning `/bin/sh` directly doesn't work too well (probably due to buffered `stdin` input), it was easiest for us to just execute an open() + read() + write() shellcode.
- Flag:

  `DrgnS{Here's_an_easy_flag_for_an_easy_pwn}`