Format String Bugs

- Format string are in the form of
  - printf(char * format, ..)
  - sprintf(char* dest, char* format, ...)
  - snprintf(char* dest, int n, char* format, ...)

- "%n": Nothing printed but writes the number of characters printed thus far to an int variable
  - E.g.
    ```c
    int x; printf("Hello %s %n", name, &x);
    %n counts the name
    ```

Printf internals /var args function

- printf(fmt, a, b);
  ```c
  fmt="%d %d"
  ```

printf() doesn’t know how many extra arguments it will have. So it initializes ARGP to print right after the format string. When it says %d, it will take bytes of ‘a’ and prints it to as int and advances to ‘b’. Interprets those bytes and prints it out. E.g

```c
void loguser(char* user)
{
    char buffer[512];
    snprintf (buffer, sizeof(buffer), user);
    .
    .
}
write(logfile, buffer,..)
```

username: “%x%x%x%x%x%x...%x%x%x”
This will print the address of stack. It can be exploited to know in the ASLR where the code in memory is. Eventually somewhere up user, loguser ret add is there and attacker can calculate the shift.
This is done to bootstrap the libc attack which has ASLR.

**Using %n to overwrite arbitrary memory**

- We can use print to write anywhere in the memory
- in the presence of ASLR and canary
- Design a format string that does
  - Mem[a] <- V
    write value V to address a
- Trick: %n
  - user = <address to be written>%n
    writes 4 to the address
  - user = <address to be written>_%n
    writes 5 to the address

- if the number is too big then we can do
  user = <address to be written>%100101d%n
  this writes an int in a spacing of 100101 bytes. %n would write 100101 at <address to be written>

- user = <a1><a2>____%n_____%n
  count=%78  count=%56

  count = 0x012345678

  In the little endian format it would be 0x78 0x56 ..

- So we can use this to inject our shell code in place of the _.
  user = <a1><a2><shell code>%n<shell code>%n