Format String Bugs

Wuftp 2.6

```c
void loguser(char *user){

    char buff(512);

    snprintf(buff,512,user);

}
```

- This prevented traditional buffer overflow attack by specifying a length to print. However, it is vulnerable to format string exploit.
- If an attacker put some special string in user
  - `%d` print the next 4 byte in the buff
  - `%s` serves as a point and direct to somewhere else
  - `%n` cause to write to some memory location the number of bytes printed so far. The location to be written is content of memory the ARGD pointer is currently pointing to
- ARGD pointer, unlike buffer pointer, will only increase by 4 byte when `%d` is read
- What we are interested is to over-write the return address using `%n`
- Supposed `0xf1234568` is the return address
  - `<0xf1234568>%n` will write 4 to address `0xf1234568`, which is the return address
  - 4 is written since only 4 byte is printed
- `%xxxxxxxxx)d` will cause the program to assume that `xxxx` byte even if there is actually nothing printed, and increase count by `xxxx`
- "00000000<0xf1234568>%86543210d%n<shell code>"
  - Value of count is 8+86543210 when %n executes
  - RA will be overwritten to 86543218 which may point to shell code which attacker supplied

TOP 25

- Integer overflow

  ```c
  int n = read(network);
  if n > 100
      Exit
  buf = malloc(n) ---BOOM when n is negative
  ```

- malloc accept unsigned integer
  - negative can be treated as a very large number this way

Command Injection Bug

- SQL injection
- CGI-BIN
  ```
  mkdir $username
  username = "rob; rm –rf /"
  ```