Format string bugs may be introduced into a program with malicious intentions by manipulating the format string buffers in a way that the return addresses are modified to point toward malicious code.

As an illustration let us consider the following example:

**Wuftp 2.6.0**

```c
void loguser(char *user)
{
    char buf[512];
    snprintf(buf,512,user);
    .
    .
    .
}
```
It must be noted that ‘n’ here refers to the count kept by printf functions (by default).

The effects of a bug in the format string can be explained as follows:

From the program it can be seen that the snprintf function can be provided with a variety of format specifiers (shown as fmt in fig(a)). If user =”%s” the address stored in the buffer is the location of the actual string. This is because all strings are stored as pointers in C. If user=“%n”, the argument pointer will store the address of the desired code implant at the address pointed to by the buffer.

For instance, if an attacker wants to plant shellcode(some malicious program) into the User space, he/she can do the as follows:

If user points to say address 86543210, the attacker will specify the user as “0000<0xf1234568>%86543210d%n<shellcode>” As a result, there is a modification of the return address of the user which is now planted with <shellcode>
**Integer Overflows**

When checks are performed for integer inputs, they must be performed at both the upper and lower bounds.

For example, consider the code snippet below:

```c
Unsigned int n=read(network)
If(n<0)
exit(0);
b=malloc(n+1);
```

Here if the value of n is the greatest upper bound on a value that is accepted by the operating system, malloc(n+1) exceeds the upper bound.

**SQL Injection Bugs**

SQL injection bugs are based on the logical manipulation of a query to get past security checks.

For example consider the following bug:

```java
void get_user(String username)
{
    String query="Select * from Users where username=\""+Username+\"\";
    db.execute(query);
}
```
If username =abcd\” or 1 is entered as input the result would be the display of all users.

A solution to this kind of an attack would be to make use of Prepared Statements.