Problem 1 Suppose that we removed the \%n format specifier from printf implementations, so that \%n no longer did anything special. (And suppose we found and fixed any programs that legitimately relied on the \%n functionality.)

In this case, would format string vulnerabilities still matter in today’s software environment? Explain.

Problem 2 For this problem, you may assume a Linux system running in a 32-bit flat memory model (i.e., with virtual addresses in the range 0x00000000–0xffffffff), with the top half of the address space (starting from 0x80000000) reserved for the kernel, and with NULL pointers of all types encoded as 0x00000000.

1. Suppose the kernel includes the following buggy system-call handling code:

```c
void handle_dangerous_syscall(void) {
    void (*fp)() = NULL; /* declare function pointer */
    (*fp)(); /* call it */
}
```

What would happen if a malicious, unprivileged program invoked the dangerous syscall?

2. Now suppose that, before invoking the dangerous syscall, the malicious, unprivileged program ran code such as:

```c
mmap(0, 4096, PROT_READ|PROT_WRITE,
     MAP_PRIVATE|MAP_ANON|MAP_FIXED, -1, 0);
```

Would this snippet allow the malicious, unprivileged program mount a more effective attack than in part (a)? Explain. Be specific.

Problem 3 Mozilla, the organization that develops Firefox, uses the Bugzilla bug-tracking system as part of its development infrastructure. Every defect and proposed enhancement in Firefox is tracked and discussed using one or more Bugzilla bug entries; some Bugzilla entries are world-readable and others can be viewed only by certain Bugzilla user accounts.

Why would an attacker want to obtain privileged Bugzilla credentials? Explain.
NAME
mmap -- allocate memory, or map files or devices into memory

LIBRARY
Standard C Library (libc, -lc)

SYNOPSIS
#include <sys/mman.h>

void *
  mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);

DESCRIPTION
The mmap() system call causes the pages starting at addr and continuing
for at most len bytes to be mapped from the object described by fd,
starting at byte offset offset. If offset or len is not a multiple of
the pagesize, the mapped region may extend past the specified range. Any
extension beyond the end of the mapped object will be zero-filled.

The addr argument is used by the system to determine the starting address
of the mapping, and its interpretation is dependent on the setting of the
MAP_FIXED flag. If MAP_FIXED is specified in flags, the system will try
to place the mapping at the specified address, possibly removing a map-
ning that already exists at that location.

The protections (region accessibility) are specified in the prot argument
by or'ing the following values:

PROT_NONE Pages may not be accessed.
PROT_READ Pages may be read.
PROT_WRITE Pages may be written.
PROT_EXEC Pages may be executed.

Note that, due to hardware limitations, on some platforms PROT_WRITE may
imply PROT_READ, and PROT_READ may imply PROT_EXEC. Portable programs
should not rely on these flags being separately enforcable.

The flags argument specifies the type of the mapped object, mapping
options and whether modifications made to the mapped copy of the page are
private to the process (copy-on-write) or are to be shared with other
references. Sharing, mapping type and options are specified in the flags
argument by or'ing the following values:

MAP_ANON Map anonymous memory not associated with any specific
file. The file descriptor is not used for creating
MAP_ANON regions, and must be specified as -1.
The mapped memory will be zero filled. The offset
argument is ignored.

MAP_FIXED Do not permit the system to select a different address
than the one specified. If the specified address can-
not be used, mmap() will fail. If MAP_FIXED is speci-
fied, addr must be a multiple of the pagesize.

MAP_PRIVATE Modifications are private (copy-on-write).

RETURN VALUES
Upon successful completion, mmap() returns a pointer to the mapped
region. Otherwise, a value of MAP_FAILED is returned and errno is set to
indicate the error.