COMP 3000: Operating Systems

Carleton University
Fall 2018 Test 2 Solutions

1. [2] If you erase a directory directly from disk (i.e., you zero out the blocks of a directory inode), are the files that were in the directory potentially recoverable? Assume none of the files had hard links elsewhere in the filesystem. Explain briefly.
   A: Yes, they are potentially recoverable. When fsck runs and finds inodes that are allocated but which have no corresponding directory entries, it will put them in lost+found.

2. [2] When connecting via SSH to access.scs.carleton.ca (which you have connected to many times) you receive a message saying **WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!** What is the "nasty" thing that could be happening? What is the more likely, benign reason for this message?
   A: The nasty thing that could be happening is an intruder-in-the-middle attack (i.e., someone could be attempting to intercept all of your communications). What is more likely is that SCS has upgraded or changed access.scs.carleton.ca in a way that caused its ssh host key to be changed.

3. [2] Where do the files (and their contents) in /sys come from? How could you create new files and directories in /sys?
   A: The contents of /sys comes from data structures in the Linux kernel. You can create files and directories by adding code to the Linux kernel that creates entries in /sys. One way to do this is by inserting a module that creates a character device.

4. [2] What does it mean when we say an executable is "setuid root"? Why does fusermount need to be setuid root in order to work?
   A: A setuid root file is an executable file that has the setuid permission bit set and is owned by root. fusermount needs to be setuid root because this allows it to run with root privileges, and root privileges are necessary to mount filesystems.

5. [2] Do programs have to do anything special to access files on a FUSE-based filesystem (that is already mounted)? Why or why not?
   A: They do not have to do anything special, files in a FUSE-based filesystem are accessed using the same system calls (open, read, write, close, etc) that are used to access regular files.

6. [2] If you run the command `truncate --size=1G foo`, how much space will `foo` take up on disk (excluding that consumed by its inode)? Will `foo` always take up this amount of space on disk, or can it be changed?
   A: foo will take up 0 blocks on disk. This amount will go up if anything is subsequently written to foo (at any position).

7. [2] Is it necessary for an mmap system call (mapping a file on disk) to immediately allocate physical memory? Why or why not?
   A: It is not necessary for an mmap system call to allocate memory immediately, it simply has to set up an area of a process's virtual memory so that when it is accessed the kernel knows to load data from the mapped file. File contents only need to be loaded from disk when the memory is accessed, not when it is mapped.

8. [2] Assume a process allocates 10GB of memory (using an anonymous mmap) on a machine with 8 GB of physical memory. Is it possible for the call to succeed? If it succeeds, what will happen as data is stored to this allocated memory? Assume that the memory is accessed *at random*.
   A: Normally the call will succeed because the Linux kernel is lazy in its memory allocations. If this memory is accessed randomly the program (and likely the whole system) will run very slowly as data is moved to and from disk—assuming there is sufficiently large swap space to store the extra data. If there isn't sufficient swap space, then eventually the system will run out of memory and it is likely processes will be killed until the kernel has enough physical and swap memory to satisfy all requests.

9. [2] Does kernel module code run as the root user? Explain briefly.
   A: Kernel module code does not run as the root user, it runs in the context of the kernel in the CPUs supervisor mode. Users (including root) are labels applied to processes, not to code running in the kernel.

10. [1] When insmod loads a kernel module, does it create a new process for the module code to run in? Explain briefly.
    A: insmod does not create a new process, it just makes the appropriate system calls in order to load the module into the kernel. The kernel then runs the module in the kernel's own execution context, not in a separate process.

11. [2] Why is copy_from_user() important when writing kernel code? Give a specific example of how you might use this in kernel code.
A: copy_from_user() is important for getting data from userspace (from a regular process). For example, when implementing a write function (say, for a character device) you would use copy_from_user() to transfer data from the calling process's buffer to a kernel data structure.

12. [2] When the Linux kernel allocates physical memory for its own data structures, does it use that memory with physical or virtual addresses? How do you know?
A: It uses that memory with a virtual address. We know this because in the remember module, after allocating physical memory we had to get its virtual address before we could use it as a character buffer.

13. [5] You attach new USB headphones to your Linux-running computer. The headphones are recognized and and seem to work properly; however, after about five minutes of listening to music, the whole system crashes, hard (keyboard death, frozen video, etc).

    (a) [2] What part of the system was likely responsible for the crash? Why?
    A: The crash was likely caused by one of the modules that were loaded when the the headphones were plugged in to the system. This code is new, it is being used, and it is running in the kernel so errors in it can cause system crashes.

    (b) [2] What is one kind of programming error that could have produced the observed behavior? Explain why it could have this impact.
    A: The module could have accessed memory improperly, say by using memory after it had been freed or writing to memory that had not been specifically allocated for the kernel module. When memory is corrupted any program can crash; when the corruption happens in the kernel the crash affects the entire system.

    (c) [1] You look online and you see that there is an alternative Linux driver for the headphones that claims to be safer because it is a "userspace driver." Why would a userspace driver potentially be safer?
    A: A userspace driver would be a driver that runs as a regular process (and communicates with the kernel using system calls). Such a driver could be safer because programming errors could at most cause the driver process to crash, rather than the entire kernel.

14. [2] In memoryll, would it be possible to change the code so all files (in the memoryll filesystem) are owned by root and executables will be marked as setuid root?

    - If this is possible, explain at a high level how you would do it. Also, explain how this doesn't lead to a situation where any user can do things as root (simply by mounting a userspace filesystem, creating a setuid root binary, and running it).
    - If not, explain how FUSE prevents mounted filesystems from having files owned by root.

    A: It is possible to make all files owned by root and executable files setuid root. All you'd need to change is the "get attributes" function that returns metadata associated with an inode (for stat-related system calls). This is not a security vulnerability because FUSE filesystems are always mounted "nosuid", meaning that the setuid and setgid bits are ignored.