Homework 3

Due: November 6, 2024 **Points:** 100

1. (*20 points*) Consider Multics procedures p and q. Procedure p is executing and needs to invoke procedure q. Procedure q's access bracket is (5, 6) and its call bracket is (6, 9). Assume that q's access control list gives p full (read, write, append, and execute) rights to q. In which ring(s) must p execute for the following to happen?

   (a) p can invoke q, but a ring-crossing fault occurs.

   (b) p can invoke q provided that a valid gate is used as an entry point.

   (c) p cannot invoke q.

   (d) p can invoke q without any ring-crossing fault occurring, but not necessarily through a valid gate.

2. (*28 points*) Most operating systems define two types of names. A *direct alias* (name or link) identifies the specific entry in a file allocation table (such as an inode), and an *indirect alias* is itself a file containing the path name of a second file. When one opens an indirect alias for certain actions (such as reading or writing), the operating system instead opens the file named in the indirect alias. Specific commands operate on the indirect alias itself (as opposed to the file it names).

   (a) Can indirect aliases ever loop; that is, can there exist a chain of indirect aliases $i_1, \ldots, i_n$ such that $i_1 = i_n$? If so, how would the system detect such loops? What should it do when one is discovered?

   (b) Can a loop with direct aliases occur?

   (c) The text points out the difference between a file name and a file descriptor. How does the introduction of indirect aliases complicate the resolution of an alias to a device number and inode?

   (d) On some systems, a direct alias cannot refer to an inode on a different device. Suppose the system were altered to allow a device number to be included in the alias, so a direct alias could refer to a file on another device. What complications might arise? Do indirect aliases, which can reference files on other devices, have the same complications?

3. (*20 points*) Classify the following vulnerabilities using the RISOS model. Assume that the classification is for the implementation level. Justify your answer.

   (a) The presence of the "wiz" command in the *sendmail* program (see Section 24.2.9).

   (b) The failure to handle the **IFS** shell variable by *loadmodule* (see Section 24.2.9).

   (c) The *heartbleed* bug occurred in an implementation of OpenSSL. A client would send a keepalive message to the server, which would echo the data of the packet back to the client. The length of the packet (including the data) was in the packet header. The first few bytes of the data also contained the length of the data, and OpenSSL used the latter to determine how big the data was. The problem was that one could give a data size of 1000 when only 5 bytes of data were present. OpenSSL would then return 1000 bytes from the input queue, revealing what information was in the buffer at those 995 bytes. (See https://xkcd.com/1354/ for an amusing description of this bug.)

   (d) The failure of the Burroughs system to detect offline changes to files (see Section 24.2.7).

4. (*22 points*) In the Janus system, when the framework disallows a system call, the error code **EINTR** (interrupted system call) is returned.

   (a) When some programs have read or write system calls terminated with this error, they retry the calls. What problems might this create?

   (b) Why did the developers of Janus not devise a new error code (say, **EJAN**) to indicate an unauthorized system call?

5. (*10 points*) The C shell does not treat the **IFS** variable as a special variable. (That is, the C shell separates arguments to commands by white spaces; this behavior is built in and cannot be changed.) How might this affect the *loadmodule* exploitation?