Your name: _____

Sign for your honor:_____

Midterm Exam
CPS 210: Operating Systems
Fall 2012

Grading the Instructor:
20 Questionable Answers to Questions You Might Have Asked

I am running short on time to grade your exam papers, so I ask you to grade mine instead. Grade the following statements on a scale of 0 (hopelessly false or garbled) to 10 (true, correct, and complete), assigning partial credit as appropriate. Write your score in the left hand margin next to the statement number. You may justify your score in the space provided below the statement, e.g., by briefly noting any exceptions or errors in the statement, or any assumptions that the statement depends on. If you do not justify your score then I might take exception with it, even if your score is justifiable. All 20 statements are equally weighted (200 points total).

In these statements I make no distinction between various versions of Unix. In particular, Linux is just a flavor of Unix. When I say Unix I am referring generally to the Unix environment as described in the reading, discussed in class, and used in your project work. When I say Android I am referring to the Android framework built above the Linux kernel.

1. In Unix systems every process is created with the fork() system call and executes with the same protection label (user ID) as its parent.

2. In both Unix and Android, Applications run in processes with separate virtual address spaces. They use the same process model implemented by the operating system kernel, but there is an important difference: in Unix, different applications running on behalf of the same user execute with the same userID, while in an Android system they run with different userIDs. That is, in Android the userID for an installed application is a property of the application, regardless of which user runs it, while in Unix it is a property of the user that runs the application, and not of the application itself: in Unix an installed application can run with different user IDs at different times, whereas in an Android system it always runs with the same userID.

3. Processes that pass data through a pipe or pipeline are children of a common parent process. They are members of the same process group, which is a different process group from the parent. The parent sets up the pipeline. Each pipe transfers data in only one direction: it links the standard output of one process with the standard input of its successor. The standard input of the process group leader receives input from the terminal. The standard output of the last process in the pipeline posts output to the terminal.

4. In Unix, one program (an initiator) can launch another (the target) only if the initiator knows the pathname of the target's executable program file. In Android, an initiator can launch a target by initiating communication with one of the target's components declared in its application manifest. It must know the name of the component, but not necessarily the name of the file(s) containing the component's code. Android finds the code and uses Unix system calls to launch the target application and set up the communication channel between the initiator and target.

5. After an attacker succeeds in mounting a trojan horse attack on a Unix user, the attacker's malware then has access to any of the user's files. But on Android a trojan horse malware program cannot access a user's data even if the user is tricked into launching the program.

6. A Unix system call occurs when a stub procedure in the standard library executes a special machine instruction that posts a trap event to the kernel, and the machine delivers it by invoking a handler routine registered by the kernel. The handler routine executes in protected kernel mode and has access to internal kernel data structures. It uses a kernel stack for the process that initiated the trap, and its execution may be interleaved with the execution of other processes on the system. When it has completed the request it returns control back to the stub that initiated the trap, executing in user mode on a stack in the process virtual memory.

7. The Unix wait() system call and variants such as waitpid() block the calling process: the process is placed in a sleep state, suspending its execution until a child process stops or exits.

8. If an attacker succeeds in compromising a Web Certifying Authority (CA), and obtaining the CA's private key, then it can create a fake version of any website. The browser HTTPS/SSL security mechanisms are unable to detect that the website is fake.

9. Secure Sockets Layer (SSL, as used in HTTPS) is based on symmetric encryption/decryption because it is faster than asymmetric encryption/decryption. This choice requires an extra step during the protocol to 'change ciphers'. Data transferred on the channel is encrypted with a key chosen by the parties for use within that channel. The key is shared by both parties, but is not known to anyone else.

Your name: _____

Sign for your honor:_____

10. A well-chosen password protects against "fake ID" attacks in which the attacker logs into a system with the identity of a victim user. The password should not be “letmein” and it should not be written down on a piece of paper next to your keyboard.

11. If Alice knows Bob's public key, and Bob signs all of his messages with his private key, then Alice can verify that a received message originated with Bob and has not been tampered. If Bob includes a nonce in each message, such as the time at which Bob sent the message, then Alice can verify that each message is fresh, i.e., that it is not an old message replayed by an attacker.

12. It is not necessary for a heap manager to zero out the memory blocks that it returns (e.g., from malloc()): the kernel zeros all virtual memory in the heap (e.g., from sbrk()).

13. A bug in an application program could cause a heap manager (Lab 1) to fail or fault, e.g., a bug in an application program could cause the heap manager code to reference a pointer that is null or wild. That could happen even if the heap manager itself is perfect. Thus the heap manager is not part of the system's Trusted Computing Base. In contrast, a bug in an application program can never cause the kernel to fail if the kernel is implemented correctly.

14. The execve() system call can pass one or more argument strings to the program that it executes, but the kernel must assist in transferring the argument data because it crosses an address space boundary for delivery into the child process. Similarly, an executed program can return a short result (an exit status) via the exit() system call, but the kernel must assist to deliver the result data across an address space boundary into the parent process.

15. A program must be careful in how it handles any data that it reads from a socket, to avoid a cross-channel attack such as a buffer/stack overflow. Support for secure sockets (e.g., SSL/HTTPS) is an important defense against such attacks.

16. A file system implementation is vulnerable to external fragmentation, but it can compact data stored on disk to reduce the problem.

17. Cryptographic hash functions (also called secure hashing or SHA) are useful even if the result digest (also called a hash or fingerprint) is not encrypted, as it is with digital signatures. For example, if Alice knows a secret, and passes Bob a digest of the secret, then Bob can determine if another party also knows the secret, even without knowing the secret himself.

18. Standard Unix file names are 'hard links', and they have a number of useful properties: a file can have multiple names/links, and the file named by a given link does not change, even if someone destroys another link/name for the file (e.g., using the unlink system call or rm command), and then reuses the same name to link to some different file with different contents. Symbolic links are different: a user can modify a symbolic link to switch it from an 'old' file to a 'new' one, and any process accessing the file through that symbolic link sees the new file rather than the old one. However, a symbolic link can be a 'dangling reference' that names no file at all, whereas a hard link always names a unique file.

19. The Garlan/Shaw paper refers to event abstractions as 'implicit invocation' because they enable the producer of an event to cause invocation of a handler procedure/method in some other component, without naming the receiving component or even having to know its name, and without invoking the method explicitly, e.g., by binding to an advertised service API and invoking it with RPC (e.g., binder) calls. The producer merely generates the event and continues executing while the event system invokes any handlers registered to receive the event. Android supports a 'broadcast' event abstraction through the 'receiver' component type.

20. Machine-level event mechanisms are important for implementing native virtual machine hypervisors. If a program generates an exception event such as a trap or fault, or if a device generates an interrupt, the program or device does not know or need to know how the event is handled, i.e., whether the OS kernel handles it or (alternatively) if it is intercepted by a hypervisor handler 'below' the kernel.