CHAPTER 16

Security

Exercises

16.1 Buffer-overflow attacks can be avoided by adopting a better programming methodology or by using special hardware support. Discuss these solutions.

Answer:
One form of hardware support that guarantees that a buffer-overflow attack does not take place is to prevent the execution of code that is located in the stack segment of a process's address space. Recall that buffer-overflow attacks are performed by overflowing the buffer on a stack frame and overwriting the return address of the function, thereby jumping to a portion of the stack frame that contains malicious executable code placed there as a result of the buffer overflow. By preventing the execution of code from the stack segment, this problem is eliminated.

Approaches that use a better programming methodology are typically built around the use of bounds checking to guard against buffer overflows. Buffer overflows do not occur in languages like Java where every array access is guaranteed to be within bounds through a software check. Such approaches require no hardware support but result in run-time costs associated with performing bounds checking.

16.2 What is the purpose of using a "salt" along with a user-provided password? Where should the salt be stored, and how should it be used?

Answer:
When a user creates a password, the system generates a random number (which is the salt) and appends it to the user-provided password. It encrypts the resulting string and stores the encrypted result and the salt in the password file. When a password check is to be made, the password presented by the user is first concatenated with the salt and then encrypted before checking for equality with the stored password. Since the salt is different for different users, a password cracker cannot check a single candidate password, encrypt it, and check it against all of the encrypted passwords simultaneously.

16.3 An experimental addition to UNIX allows a user to connect a **watchdog** program to a file. The watchdog is invoked whenever a program requests access to the file. The watchdog then either grants or denies access to the file. Discuss two pros and two cons of using watchdogs for security.

Answer:
The watchdog program is the primary security mechanism for file access. A benefit of this approach is that it provides a centralized mechanism for controlling access to a file—the watchdog program. If the watchdog program's security techniques are sufficient, you are assured of having secure access to the file. Also, watchdogs allow many more kinds of access protection to files. However, centralization is also the primary disadvantage of this approach —the watchdog program becomes the bottleneck. If the watchdog program is not properly implemented (that is, if it has a security hole), there are no backup mechanisms for file protection.

16.4 Make a list of six security concerns for a bank's computer system. For each item on your list, state whether this concern relates to physical, human, or operating-system security.

Answer:

- In a protected location, well guarded: physical, human
- Network tamperproof: physical, human, operating system
- Modem access eliminated or limited: physical, human
- Unauthorized data transfers prevented or logged: human, operating system
- Backup media protected and guarded: physical, human
- Programmers, data entry personnel, trustworthy: human

16.5 What commonly used computer programs are prone to man-in-the-middle attacks? Discuss solutions for preventing this form of attack.

Answer:
Any protocol that requires a sender and a receiver to agree on a session key before they start communicating is prone to the man-in-the-middle attack. For instance, if we implement a secure shell protocol by having the two communicating machines identify a common session key, and if the protocol message for exchanging the session key is not protected by the appropriate authentication mechanism, then it is possible for an attacker to manufacture a separate session key and get access to the data being communicated between the two parties. In particular, if the server is supposed to manufacture the session key, the attacker can obtain the session key from the server, send its own session key to the client, and convince the client to use the fake session key. When the attacker receives the data from the client, it can decrypt the data, reencrypt the data with the original key from the server, and transmit the encrypted data to the server without alerting either the client or the server to the attack. Such attacks could be avoided by using digital

signatures to authenticate messages from the server. If the server could communicate the session key and its identity in a message guarded by a digital signature granted by a certifying authority, then the attacker would not be able to forge a session key, and the man-in-the-middle attack could be avoided.

16.6 Why doesn't $D_{kd,N}(E_{ke,N}(m))$ provide authentication of the sender? To what uses can such an encryption be put?

Answer:
$D(k_d, N)(E(k_e, N)(m))$ means that the message is encrypted using the public key and then decrypted using the private key. This scheme is not sufficient to guarantee authentication, since any entity could have obtained the public keys and therefore could have fabricated the message. However, the only entity that can decrypt the message is the entity that owns the private key, which guarantees that the message is a secret message from the sender to the entity owning the private key; no other entity can decrypt the contents of the message.

16.7 Mobile operating systems such as iOS and Android place the user data and the system files into two separate partitions. Aside from security, what is an advantage of that separation?

Answer:
User data can be encrypted easily (encrypt the user file system but not the system file system), user data can be erased without harming system (reset to factory defaults), and the system can be upgraded without harming user data.