SOLUTIONS MANUAL

COMPUTER SECURITY THIRD EDITION

CHAPTERS 1–12

WILLIAM STALLINGS
LAWRIE BROWN

NOTICE

This manual contains solutions to the review questions and homework problems in *Computer Security, Third Edition*. If you spot an error in a solution or in the wording of a problem, I would greatly appreciate it if you would forward the information via email to wllmst@me.net. An errata sheet for this manual, if needed, is available at http://www.box.net/shared/ds8lygu0tjljokf98k85 . File name is S-CompSec3e-mmyy.

TABLE OF CONTENTS

CHAPTER 1 OVERVIEW

ANSWERS TO QUESTIONS

1.1 The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).

1.2 The OSI Security Architecture is a framework that provides a systematic way of defining the requirements for security and characterizing the approaches to satisfying those requirements. The document defines security attacks, mechanisms, and services, and the relationships among these categories.

1.3 Passive attacks have to do with eavesdropping on, or monitoring, transmissions. Electronic mail, file transfers, and client/server exchanges are examples of transmissions that can be monitored. Active attacks include the modification of transmitted data and attempts to gain unauthorized access to computer systems.

1.4 Passive attacks: release of message contents and traffic analysis. Active attacks: masquerade, replay, modification of messages, and denial of service.

1.5 **Authentication:** The assurance that the communicating entity is the one that it claims to be.
Access control: The prevention of unauthorized use of a resource (i.e., this service controls who can have access to a resource, under what conditions access can occur, and what those accessing the resource are allowed to do).
Data confidentiality: The protection of data from unauthorized disclosure.
Data integrity: The assurance that data received are exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion, or replay).
Nonrepudiation: Provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication.

Availability service: The property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system (i.e., a system is available if it provides services according to the system design whenever users request them).

1.6 See Table 1.6.

ANSWERS TO PROBLEMS

1.1 The system must keep personal identification numbers confidential, both in the host system and during transmission for a transaction. It must protect the integrity of account records and of individual transactions. Availability of the host system is important to the economic well being of the bank, but not to its fiduciary responsibility. The availability of individual teller machines is of less concern. Example from [NRC91].

1.2 The system does not have high requirements for integrity on individual transactions, as lasting damage will not be incurred by occasionally losing a call or billing record. The integrity of control programs and configuration records, however, is critical. Without these, the switching function would be defeated and the most important attribute of all - availability - would be compromised. A telephone switching system must also preserve the confidentiality of individual calls, preventing one caller from overhearing another. Example from [NRC91].

1.3 **a.** The system will have to assure confidentiality if it is being used to publish corporate proprietary material.
b. The system will have to assure integrity if it is being used to laws or regulations.
c. The system will have to assure availability if it is being used to publish a daily paper. Example from [NRC91].

1.4 **a.** An organization managing public information on its web server determines that there is no potential impact from a loss of confidentiality (i.e., confidentiality requirements are not applicable), a moderate potential impact from a loss of integrity, and a moderate potential impact from a loss of availability.
b. A law enforcement organization managing extremely sensitive investigative information determines that the potential impact from a loss of confidentiality is high, the potential impact from a loss of integrity is moderate, and the potential impact from a loss of availability is moderate.
c. A financial organization managing routine administrative information (not privacy-related information) determines that the potential

impact from a loss of confidentiality is low, the potential impact from a loss of integrity is low, and the potential impact from a loss of availability is low.

d. The management within the contracting organization determines that: (i) for the sensitive contract information, the potential impact from a loss of confidentiality is moderate, the potential impact from a loss of integrity is moderate, and the potential impact from a loss of availability is low; and (ii) for the routine administrative information (non-privacy-related information), the potential impact from a loss of confidentiality is low, the potential impact from a loss of integrity is low, and the potential impact from a loss of availability is low.

e. The management at the power plant determines that: (i) for the sensor data being acquired by the SCADA system, there is no potential impact from a loss of confidentiality, a high potential impact from a loss of integrity, and a high potential impact from a loss of availability; and (ii) for the administrative information being processed by the system, there is a low potential impact from a loss of confidentiality, a low potential impact from a loss of integrity, and a low potential impact from a loss of availability. Examples from FIPS 199.

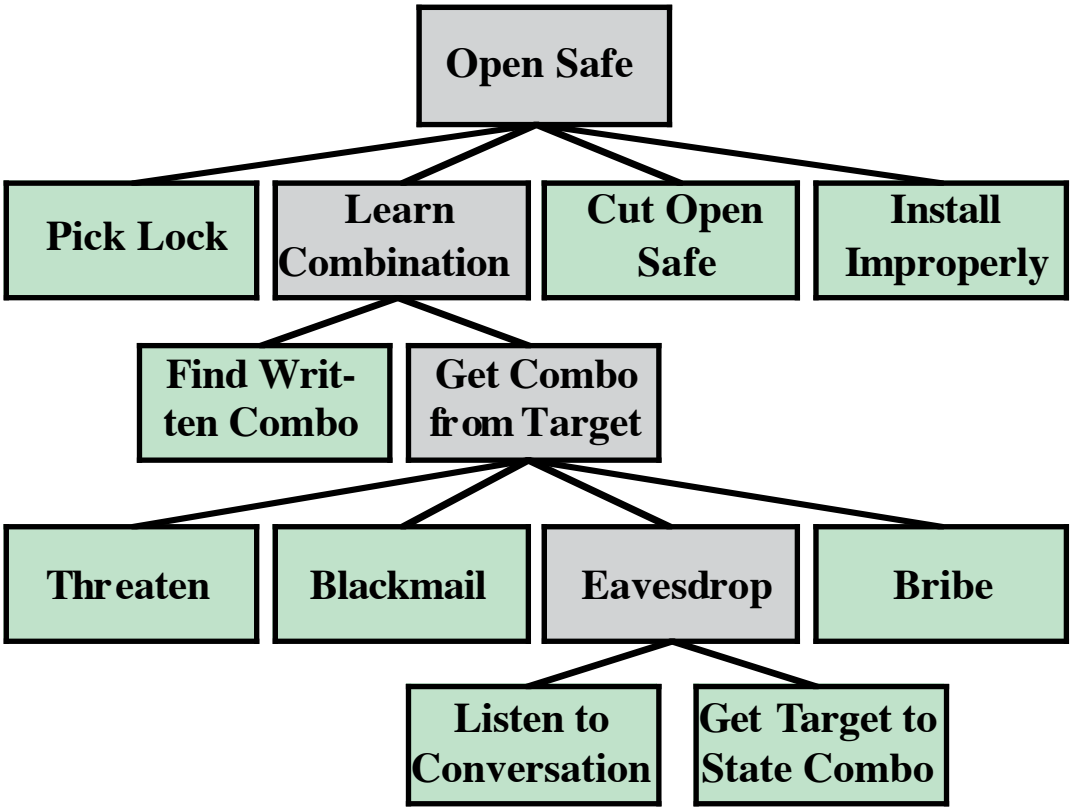
1.5 a. At first glance, this code looks fine, but what happens if IsAccessAllowed fails? For example, what happens if the system runs out of memory, or object handles, when this function is called? The user can execute the privileged task because the function might return an error such as ERROR NOT ENOUGH MEMORY.

b. x

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
    // Secure check OK.
    // Perform task.
} else {
    // Security check failed.
    // Inform user that access is denied.
}
```

In this case, if the call to IsAccessAllowed fails for any reason, the user is denied access to the privileged operation.

1.6



1.7 We present the tree in text form; call the company X:

Survivability Compromise: Disclosure of X proprietary secrets

OR 1. Physically scavenge discarded items from X
- OR 1. Inspect dumpster content on-site
- 2. Inspect refuse after removal from site

2. Monitor emanations from X machines
- AND 1. Survey physical perimeter to determine optimal monitoring position
- 2. Acquire necessary monitoring equipment
- 3. Setup monitoring site
- 4. Monitor emanations from site

3. Recruit help of trusted X insider
- OR 1. Plant spy as trusted insider
- 2. Use existing trusted insider

4. Physically access X networks or machines
- OR 1. Get physical, on-site access to Intranet
- 2. Get physical access to external machines

5. Attack X intranet using its connections with Internet
- OR 1. Monitor communications over Internet for leakage
- 2. Get trusted process to send sensitive information to attacker over Internet
- 3. Gain privileged access to Web server

6. Attack X intranet using its connections with public telephone network (PTN)
- OR 1. Monitor communications over PTN for leakage of sensitive information
- 2. Gain privileged access to machines on intranet connected via Internet

CHAPTER 2 CRYPTOGRAPHIC TOOLS

ANSWERS TO QUESTIONS

2.1 Plaintext, encryption algorithm, secret key, ciphertext, decryption algorithm.

2.2 One secret key.

2.3 (1) a strong encryption algorithm; (2) Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure.

2.4 Message encryption, message authentication code, hash function.

2.5 An authenticator that is a cryptographic function of both the data to be authenticated and a secret key.

2.6 **(a)** A hash code is computed from the source message, encrypted using symmetric encryption and a secret key, and appended to the message. At the receiver, the same hash code is computed. The incoming code is decrypted using the same key and compared with the computed hash code. **(b)** This is the same procedure as in (a) except that public-key encryption is used; the sender encrypts the hash code with the sender's private key, and the receiver decrypts the hash code with the sender's public key. **(c)** A secret value is appended to a message and then a hash code is calculated using the message plus secret value as input. Then the message (without the secret value) and the hash code are transmitted. The receiver appends the same secret value to the message and computes the hash value over the message plus secret value. This is then compared to the received hash code.

2.7

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h.
5. For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x).

6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).

2.8 **Plaintext:** This is the readable message or data that is fed into the algorithm as input. **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext. **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input. **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts. **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

2.9 **Encryption/decryption:** The sender encrypts a message with the recipient's public key. **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message. **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

2.10 The key used in conventional encryption is typically referred to as a **secret key**. The two keys used for public-key encryption are referred to as the **public key** and the **private key**.

2.11 A **digital signature** is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.

2.12 A **pubic-key certificate** consists of a public key plus a User ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA) that is trusted by the user community, such as a government agency or a financial institution.

2.13 Several different approaches are possible, involving the private key(s) of one or both parties. One approach is Diffie-Hellman key exchange. Another approach is for the sender to encrypt a secret key with the recipient's public key.

ANSWERS TO PROBLEMS

2.1 Yes. The eavesdropper is left with two strings, one sent in each direction, and their XOR is the secret key.

2.2 a.

2	8	10	7	9	6	3	1	4	5
C	R	Y	P	T	O	G	A	H	I
B	E	A	T	T	H	E	T	H	I
R	D	P	I	L	L	A	R	F	R
O	M	T	H	E	L	E	F	T	O
U	T	S	I	D	E	T	H	E	L
Y	C	E	U	M	T	H	E	A	T
R	E	T	O	N	I	G	H	T	A
T	S	E	V	E	N	I	F	Y	O
U	A	R	E	D	I	S	T	R	U
S	T	F	U	L	B	R	I	N	G
T	W	O	F	R	I	E	N	D	S

4	2	8	10	5	6	3	7	1	9
N	E	T	W	O	R	K	S	C	U
T	R	F	H	E	H	F	T	I	N
B	R	O	U	Y	R	T	U	S	T
E	A	E	T	H	G	I	S	R	E
H	F	T	E	A	T	Y	R	N	D
I	R	O	L	T	A	O	U	G	S
H	L	L	E	T	I	N	I	B	I
T	I	H	I	U	O	V	E	U	F
E	D	M	T	C	E	S	A	T	W
T	L	E	D	M	N	E	D	L	R
A	P	T	S	E	T	E	R	F	O

```
ISRNG  BUTLF  RRAFR  LIDLP  FTIYO  NVSEE  TBEHI  HTETA
EYHAT  TUCME  HRGTA  IOENT  TUSRU  IEADR  FOETO  LHMET
NTEDS  IFWRO  HUTEL  EITDS
```

b. The two matrices are used in reverse order. First, the ciphertext is laid out in columns in the second matrix, taking into account the order dictated by the second memory word. Then, the contents of the second matrix are read left to right, top to bottom and laid out in columns in the first matrix, taking into account the order dictated by the first memory word. The plaintext is then read left to right, top to bottom.

c. Although this is a weak method, it may have use with time-sensitive information and an adversary without immediate access to good cryptanalysis t(e.g., tactical use). Plus it doesn't require anything more than paper and pencil, and can be easily remembered.

2.3 a. Let -X be the additive inverse of X. That is -X $\boxed{+}$ X = 0. Then:

$$P = (C \boxed{+} -K_1) \oplus K_0$$

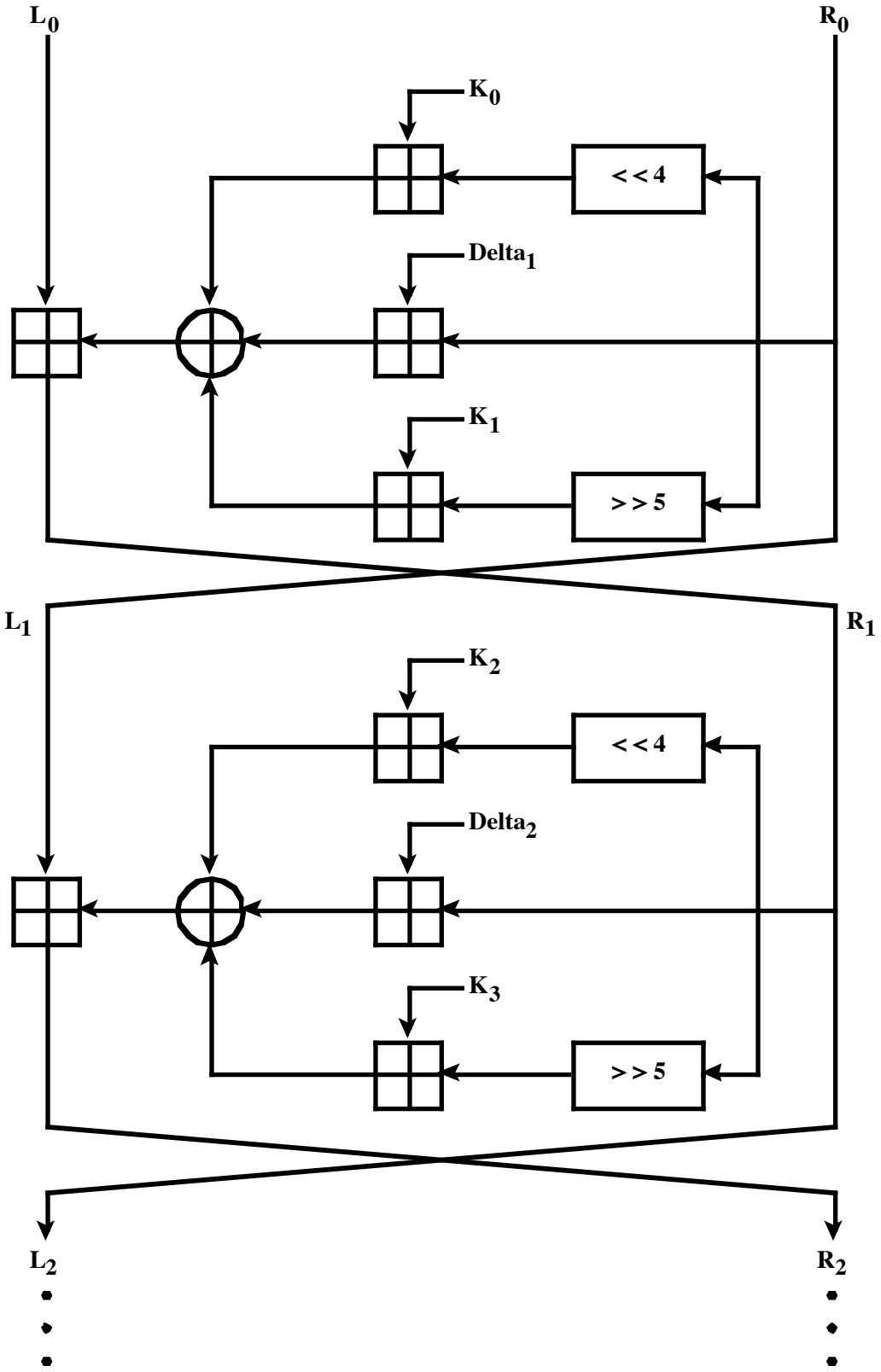
b. First, calculate –C'. Then –C' = (P' $\oplus$ K_0) $\boxed{+}$ (– K_1). We then have:

C $\boxed{+}$ –C' = (P $\oplus$ K_0) $\boxed{+}$ (P' $\oplus$ K_0)

However, the operations $\boxed{+}$ and $\oplus$ are not associative or distributive with one another, so it is not possible to solve this equation for K_0.

2.4 a. The constants ensure that encryption/decryption in each round is different.

b. First two rounds:



c. First, let's define the encryption process:

$L_2 = L_0 \boxed{+} [(R_0 << 4) \boxed{+} K_0] \oplus [R_0 \boxed{+} \delta_1] \oplus [(R_0 >> 5) \boxed{+} K_1]$

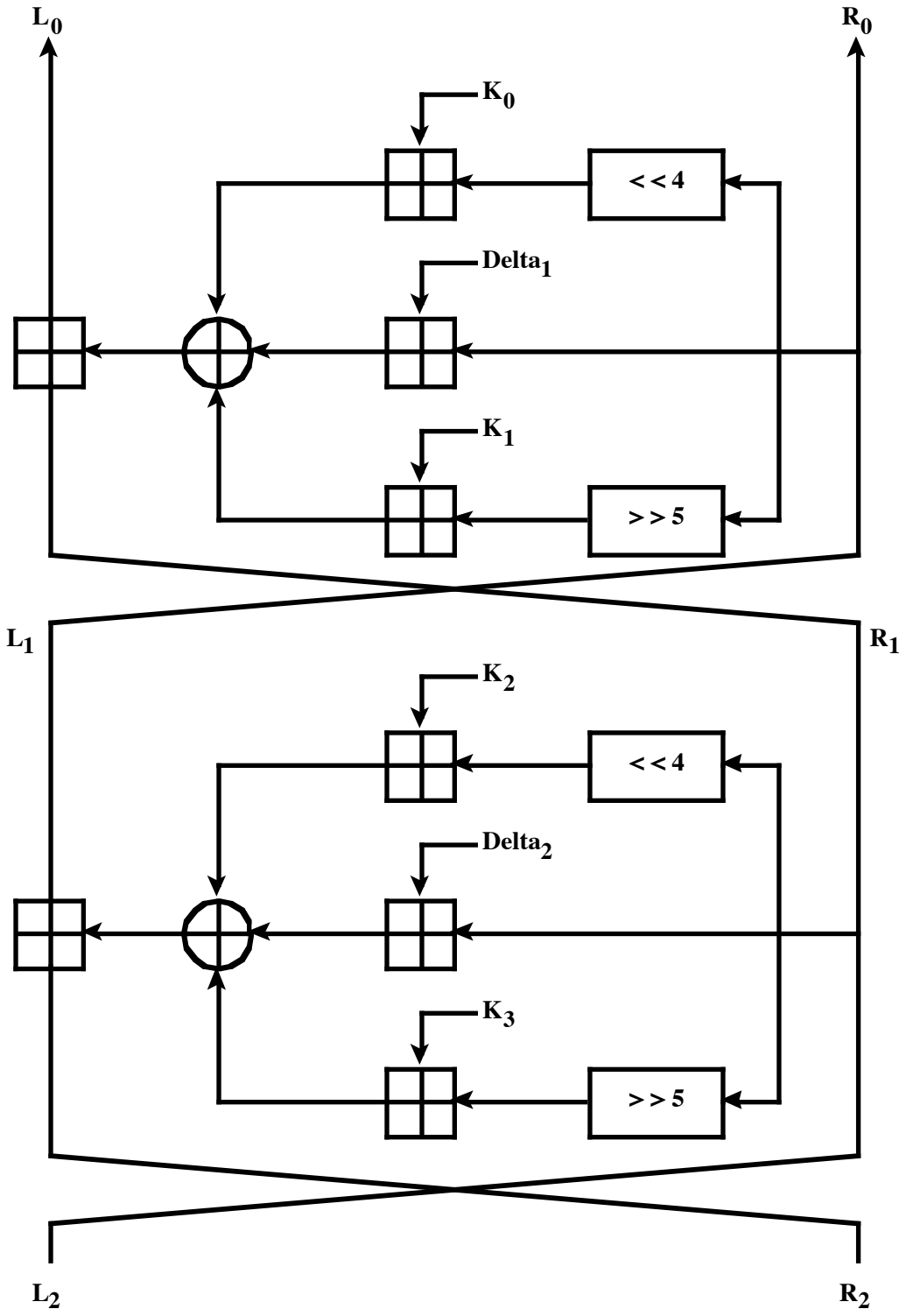
$R_2 = R_0 \boxed{+} [(L_2 << 4) \boxed{+} K_2] \oplus [L_2 \boxed{+} \delta_2] \oplus [(L_2 >> 5) \boxed{+} K_3]$

Now the decryption process. The input is the ciphertext (L_2, R_2), and the output is the plaintext (L_0, R_0). Decryption is essentially the same as encryption, with the subkeys and delta values applied in reverse order. Also note that it is not necessary to use subtraction because there is an even number of additions in each equation.

$R_0 = R_2 \boxed{+} [(L_2 << 4) \boxed{+} K_2] \oplus [L_2 \boxed{+} \delta_2] \oplus [(L_2 >> 5) \boxed{+} K_3]$

$L_0 = L_2 \boxed{+} [(R_0 << 4) \boxed{+} K_0] \oplus [R_0 \boxed{+} \delta_1] \oplus [(R_0 >> 5) \boxed{+} K_1]$

d.



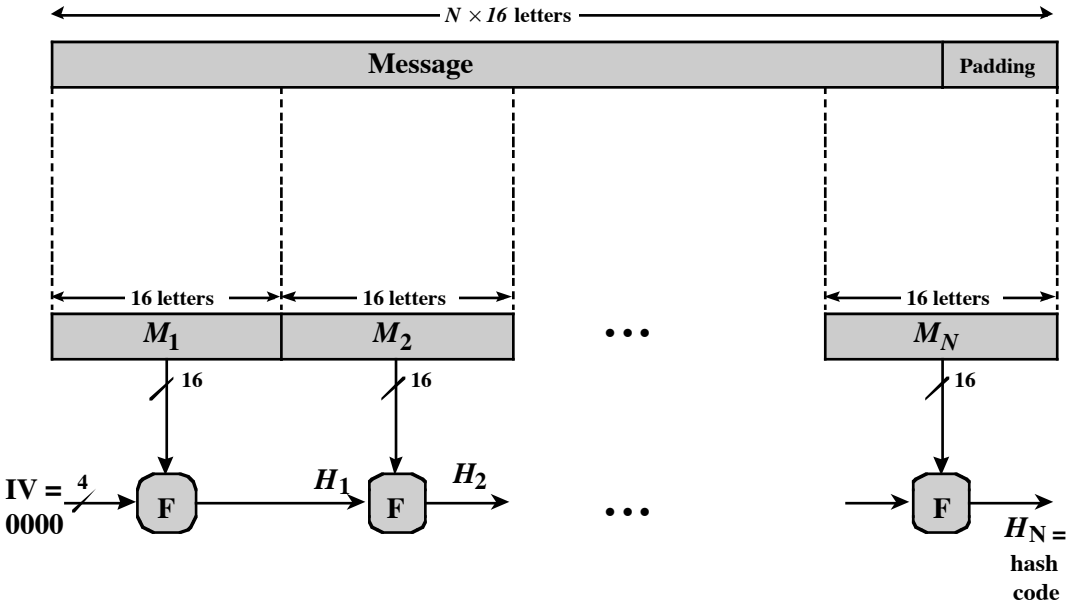
2.5 **a.** Will be detected with both (i) DS and (ii) MAC.
b. Won't be detected by either (Remark: use timestamps).
c. (i) DS: Bob simply has to verify the message with the public key from both. Obviously, only Alice's public key results in a successful verification.

(ii) MAC: Bob has to challenge both, Oscar and Bob, to reveal their secret key to him (which he knows anyway). Only Bob can do that.

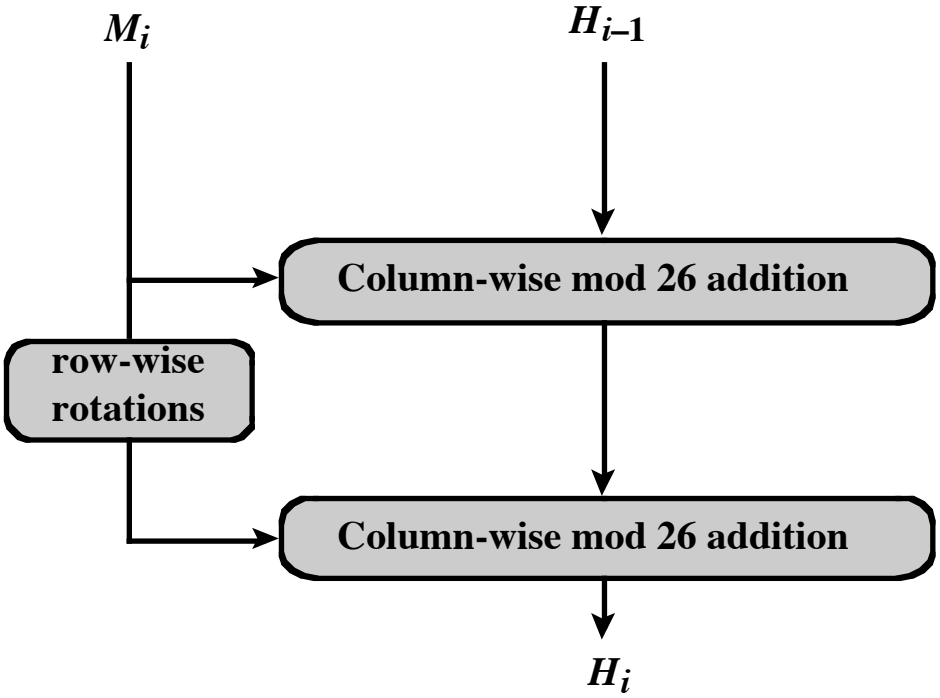
d. (i) DS: Alice has to force Bob to prove his claim by sending her a copy of the message in question with the signature. Then Alice can show that message and signature can be verified with Bob's public key) Bob must have generated the message.
(ii) MAC: No, Bob can claim that Alice generated this message.

2.6 The statement is false. Such a function cannot be one-to-one because the number of inputs to the function is of arbitrary, but the number of unique outputs is 2^n. Thus, there are multiple inputs that map into the same output.

2.7 a. Overall structure:



Compression function F:



b. BFQG

c. Simple algebra is all you need to generate a result:

AYHGDAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAA

2.8

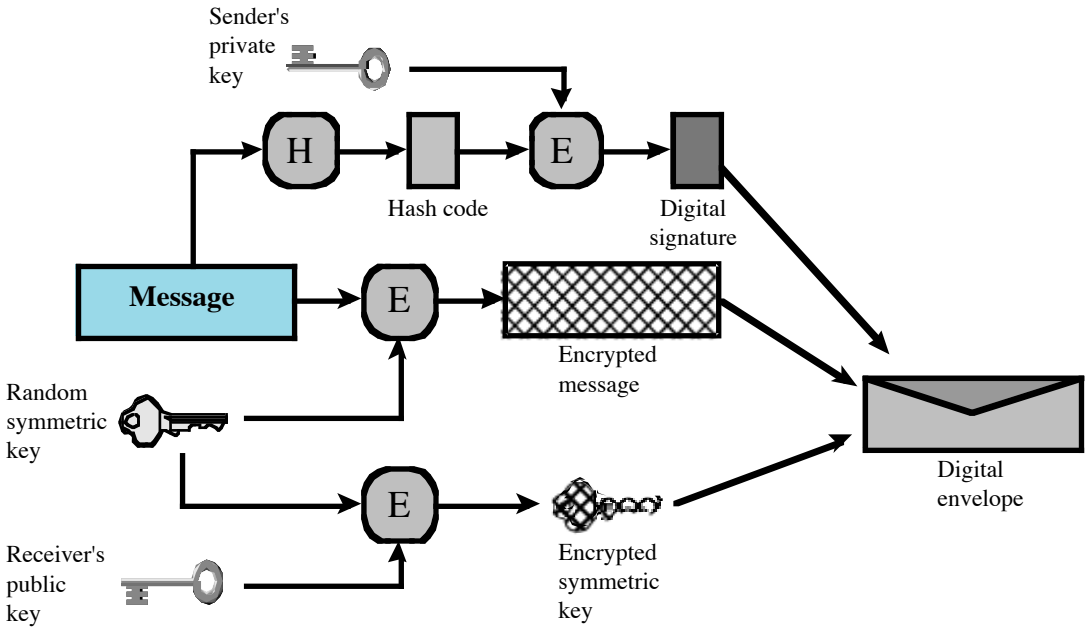
a. $M3 =$

5	2	1	4	5
1	4	3	2	2
3	1	2	5	3
4	3	4	1	4
2	5	5	3	1

b. Assume a plaintext message p is to be encrypted by Alice and sent to Bob. Bob makes use of M1 and M3, and Alice makes use of M2. Bob chooses a random number, k, as his private key, and maps k by M1 to get x, which he sends as his public key to Alice. Alice uses x to encrypt p with M2 to get z, the ciphertext, which she sends to Bob. Bob uses k to decrypt z by means of M3, yielding the plaintext message p.

c. If the numbers are large enough, and M1 and M2 are sufficiently random to make it impractical to work backwards, p cannot be found without knowing k.

2.9 We show the creation of a digital envelope:



CHAPTER 3 USER AUTHENTICATION

ANSWERS TO QUESTIONS

3.1 **Something the individual knows:** Examples includes a password, a personal identification number (PIN), or answers to a prearranged set of questions.
Something the individual possesses: Examples include electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a token.
Something the individual is (static biometrics): Examples include recognition by fingerprint, retina, and face.
Something the individual does (dynamic biometrics): Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

3.2 We can identify the following attack strategies and countermeasures:
Offline dictionary attack: Typically, strong access controls are used to protect the system's password file. However, experience shows that determined hackers can frequently bypass such controls and gain access to the file. The attacker obtains the system password file and compares the password hashes against hashes of commonly used passwords. If a match is found, the attacker can gain access by that ID/password combination.
Specific account attack: The attacker targets a specific account and submits password guesses until the correct password is discovered.
Popular password attack: A variation of the preceding attack is to use a popular password and try it against a wide range of user IDs. A user's tendency is to choose a password that is easily remembered; this unfortunately makes the password easy to guess.
Password guessing against single user: The attacker attempts to gain knowledge about the account holder and system password policies and uses that knowledge to guess the password.
Workstation hijacking: The attacker waits until a logged-in workstation is unattended.
Exploiting user mistakes: If the system assigns a password, then the user is more likely to write it down because it is difficult to remember. This situation creates the potential for an adversary to read the written password. A user may intentionally share a password, to enable a colleague to share files, for example. Also, attackers are frequently

successful in obtaining passwords by using social engineering tactics that trick the user or an account manager into revealing a password. Many computer systems are shipped with preconfigured passwords for system administrators. Unless these preconfigured passwords are changed, they are easily guessed.

Exploiting multiple password use. Attacks can also become much more effective or damaging if different network devices share the same or a similar password for a given user.

Electronic monitoring: If a password is communicated across a network to log on to a remote system, it is vulnerable to eavesdropping. Simple encryption will not fix this problem, because the encrypted password is, in effect, the password and can be observed and reused by an adversary.

3.3 One technique is to restrict access to the password file using standard access control measures. Another technique is to force users to select passwords that are difficult to guess.

3.4 **User education:** Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords.

Computer-generated passwords: the system selects a password for the user.

Reactive password checking: the system periodically runs its own password cracker to find guessable passwords.

Proactive password checking: a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it.

3.5 Memory cards can store but not process data. Smart cards have a microprocessor.

3.6 **Facial characteristics:** Facial characteristics are the most common means of human-to-human identification; thus it is natural to consider them for identification by computer. The most common approach is to define characteristics based on relative location and shape of key facial features, such as eyes, eyebrows, nose, lips, and chin shape. An alternative approach is to use an infrared camera to produce a face thermogram that correlates with the underlying vascular system in the human face.

Fingerprints: Fingerprints have been used as a means of identification for centuries, and the process has been systematized and automated particularly for law enforcement purposes. A fingerprint is the pattern of ridges and furrows on the surface of the fingertip. Fingerprints are believed to be unique across the entire human population. In practice, automated fingerprint recognition and matching system extract a

number of features from the fingerprint for storage as a numerical surrogate for the full fingerprint pattern.

Hand geometry: Hand geometry systems identify features of the hand, including shape, and lengths and widths of fingers.

Retinal pattern: The pattern formed by veins beneath the retinal surface is unique and therefore suitable for identification. A retinal biometric system obtains a digital image of the retinal pattern by projecting a low-intensity beam of visual or infrared light into the eye.

Iris: Another unique physical characteristic is the detailed structure of the iris.

Signature: Each individual has a unique style of handwriting and this is reflected especially in the signature, which is typically a frequently written sequence. However, multiple signature samples from a single individual will not be identical. This complicates the task of developing a computer representation of the signature that can be matched to future samples.

Voice: Whereas the signature style of an individual reflects not only the unique physical attributes of the writer but also the writing habit that has developed, voice patterns are more closely tied to the physical and anatomical characteristics of the speaker. Nevertheless, there is still a variation from sample to sample over time from the same speaker, complicating the biometric recognition task.

3.7 **Enrollment** is analogous to assigning a password to a user. For a biometric system, the user presents a name and, typically, some type of password or PIN to the system. At the same time the system senses some biometric characteristic of this user (e.g., fingerprint of right index finger). The system digitizes the input and then extracts a set of features that can be stored as a number or set of numbers representing this unique biometric characteristic; this set of numbers is referred to as the user's template. The user is now enrolled in the system, which maintains for the user a name (ID), perhaps a PIN or password, and the biometric value. **Verification** is analogous to a user logging on to a system by using a memory card or smart card coupled with a password or PIN. For biometric verification, the user enters a PIN and also uses a biometric sensor. The system extracts the corresponding feature and compares that to the template stored for this user. If there is a match, then the system authenticates this user.

For an **identification** system, the individual uses the biometric sensor but presents no additional information. The system then compares the presented template with the set of stored templates. If there is a match, then this user is identified. Otherwise, the user is rejected

3.8 A false match occurs when an imposter's biometric data is declared by the system to be matched with the stored biometric data for a user. A false mismatch occurs when the system declares that the biometric data

of a genuine user does not match the stored biometric data for that user. The rate refers to the probability of a false match or false mismatch.

3.9 In general terms, a challenge-response protocol functions as follows. A user attempts to logon to a server. The server issues some sort of challenge that the user must respond to in order to be authenticated.

ANSWERS TO PROBLEMS

3.1 **a.** If this is a license plate number, that is easily guessable.
b. suitable
c. easily guessable
d. easily guessable
e. easily guessable
f. suitable
g. very unsuitable
h. This is bigbird in reverse; not suitable.

3.2 The number of possible character strings of length 8 using a 36-character alphabet is $36^8 \approx 2^{41}$. However, only 2^{15} of them need be looked at, because that is the number of possible outputs of the random number generator. This scheme is discussed in [MORR79].

3.3 **a.** $T = \frac{26^4}{2}$ seconds = 63.5 hours

b. Expect 13 tries for each digit. $T = 13 \times 4 = 52$ seconds.

3.4 **a.** $p = r^k$

b. $p = \frac{r^k - r^p}{r^{k+p}}$

c. $p = r^p$

3.5 **a.** $T = (21 \times 5 \times 21)^2 = 4{,}862{,}025$
b. $p = 1/T \approx 2 \times 10^{-7}$

3.6 There are $95^{10} \approx 6 \times 10^{19}$ possible passwords. The time required is:

$$\frac{6 \times 10^{19} \text{ passwords}}{6.4 \times 10^{6} \text{ passwords/second}} = 9.4 \times 10^{12} \text{ seconds}$$

$$= 300{,}000 \text{ years}$$

3.7 a. Since PU_a and PR_a are inverses, the value PR_a can be checked to validate that P_a was correctly supplied: Simply take some arbitrary block X and verify that X = D(PRa, E[PUa, X]).

b. Since the file /etc/publickey is publicly readable, an attacker can guess P (say P') and compute $PR_{a'}$ = D(P', E[P, PR_a]). now he can choose an arbitrary block Y and check to see if Y = D(PR_a, E[PU_a, Y]). If so, it is highly probable that P' = P. Additional blocks can be used to verify the equality.

3.8 Without the salt, the attacker can guess a password and encrypt it. If ANY of the users on a system use that password, then there will be a match. With the salt, the attacker must guess a password and then encrypt it once for each user, using the particular salt for each user.

3.9 It depends on the size of the user population, not the size of the salt, since the attacker presumably has access to the salt for each user. The benefit of larger salts is that the larger the salt, the less likely it is that two users will have the same salt. If multiple users have the same salt, then the attacker can do one encryption per password guess to test all of those users.

3.10 a. If there is only one hash function (k = 1), which produces one of N possible hash values, and there is only one word in the dictionary, then the probability that an arbitrary bit b_i is set to 1 is just 1/N. If there are k hash functions, let us assume for simplicity that they produce k distinct hash functions for a given word. This assumption only introduces a small margin of error. Then, the probability that an arbitrary bit b_i is set to 1 is k/N. Therefore, the probability that b_i is equal to 0 is 1 – k/N. The probability that a bit is left unset after D dictionary words are processed is just the probability that each of the D transformations set other bits:

$$\Pr[b_i = 0] = \left(1 - \frac{k}{N}\right)^D$$

This can also be interpreted as the expected fraction of bits that are equal to 0.

b. A word not in the dictionary will be falsely accepted if all k bits tested are equal to 1. Now, from part (a), we can say that the expected fraction of bits in the hash table that are equal to one is $1 - \phi$. The probability that a random word will be mapped by a single hash function onto a bit that is already set is the probability that the bit generated by the hash function is in the set of bits equal to one, which is just $1 - \phi$. Therefore, the probability that the k hash functions applied to the word will produce k bits all of which are in the set of bits equal to one is $(1 - \phi)^k$.

c. We use the approximation $(1 - x) \approx e^{-x}$.

3.11 For a static biometric device, the user's biometric is matched to a template at the client, thus the client biometric device must be authenticate. For a dynamic biometric device, the host or server generates a random sequence of numbers, characters, or words. The user then generates a response (handwriting, voice) and the result is analyzed at the host. In this case, the client biometric device need not be authenticated.

CHAPTER 4 ACCESS CONTROL

ANSWERS TO QUESTIONS

4.1 **Discretionary access control (DAC)** controls access based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do. This policy is termed *discretionary* because an entity might have access rights that permit the entity, by its own volition, to enable another entity to access some resource. **Mandatory access control (MAC)** controls access based on comparing security labels (which indicate how sensitive or critical system resources are) with security clearances (which indicate system entities are eligible to access certain resources). This policy is termed *mandatory* because an entity that has clearance to access a resource may not, just by its own volition, enable another entity to access that resource.

4.2 **Role-based access control (RBAC)** controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles. RBAC may have a discretionary or mandatory mechanism.

4.3 **Owner:** This may be the creator of a resource, such as a file. For system resources, ownership may belong to a system administrator. For project resources, a project administrator or leader may be assigned ownership.
Group: In addition to the privileges assigned to an owner, a named group of users may also be granted access rights, such that membership in the group is sufficient to exercise these access rights. In most schemes, a user may belong to multiple groups.
World: The least amount of access is granted to users who are able to access the system but are not included in the categories owner and group for this resource.

4.4 A **subject** is an entity capable of accessing objects. Generally, the concept of subject equates with that of process. Any user or application actually gains access to an object by means of a process that represents that user or application. An **object** is anything to which access is controlled. Examples include files, portions of files, programs, and segments of memory.

4.5 An **access right** describes the way in which a subject may access an object.

4.6 For each object, an **access control list** lists users and their permitted access rights. A **capability ticket** specifies authorized objects and operations for a user.

4.7 A protection domain is a set of objects together with access rights to those objects.

4.8 $RBAC_0$ contains the minimum functionality for an RBAC system. $RBAC_1$ includes the $RBAC_0$ functionality and adds role hierarchies, which enable one role to inherit permissions from another role. $RBAC_2$ includes $RBAC_0$ and adds constraints, which restrict the ways in which the components of a RBAC system may be configured. $RBAC_3$ contains the functionality of $RBAC_0$, $RBAC_1$, and $RBAC_2$.

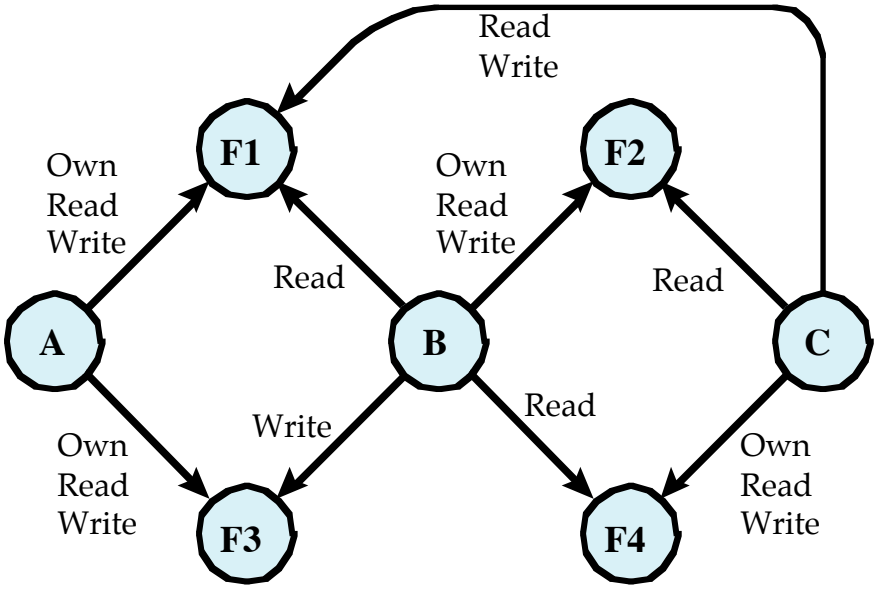
4.9 **User:** An individual that has access to this computer system. Each individual has an associated user ID.
Role: A named job function within the organization that controls this computer system. Typically, associated with each role is a description of the authority and responsibility conferred on this role, and on any user who assumes this role.
Permission: An approval of a particular mode of access to one or more objects. Equivalent terms are access right, privilege, and authorization.
Session: A mapping between a user and an activated subset of the set of roles to which the user is assigned.

4.10 **Mutually exclusive roles** are roles such that a user can be assigned to only one role in the set. **Cardinality** refers to setting a maximum number with respect to roles. A system might be able to specify a **prerequisite**, which dictates that a user can only be assigned to a particular role if it is already assigned to some other specified role.

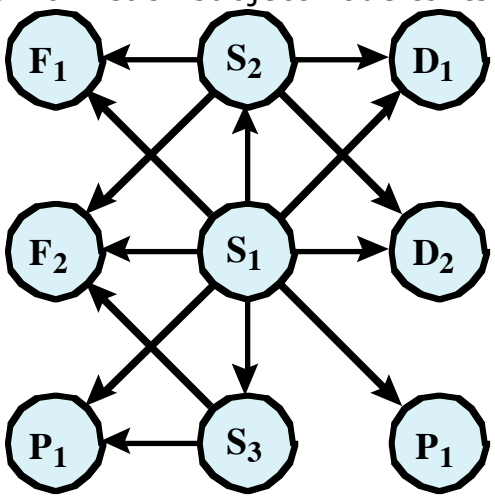
4.11 SSD enables the definition of a set of mutually exclusive roles, such that if a user is assigned to one role in the set, the user may not be assigned to any other role in the set. DSD specifications limit the availability of the permissions by placing constraints on the roles that can be activated within or across a user's sessions.

ANSWERS TO PROBLEMS

4.1 a.



b. For simplicity and clarity, the labels are omitted. Also, there should be arrowed lines from each subject node to itself.



c. A given access matrix generates only one directed graph, and a given directed graph yields only one access matrix, so the correspondence is one-to-one.

4.2 a. We could implement "subject" objects that are not associated with any user. The subject would be a protection domain and would have all the privileges that subject has in the access control lists. An operation on the subject object would be to allow the process to 'become" that subject. Being that subject would be equivalent to being in that protection domain.

b. It is only necessary to change the system so that sets of capabilities can exist independent of a process, and a process can run in a protection domain. There would be a capability to enter each

protection domain, and only processes holding that capability would be allowed to enter. You could have some capabilities associated with the process and some with the protection domain it is running in.

4.3 **a.** The advantage of four modes is that there is more flexibility to control access to memory, allowing finer tuning of memory protection. The disadvantage is complexity and processing overhead. For example, procedures running at each of the access modes require separate stacks with appropriate accessibility.

b. In principle, the more modes, the more flexibility, but it seems difficult to justify going beyond four.

4.4 **a.** With $j < i$, a process running in D_i is prevented from accessing objects in D_j. Thus, if D_j contains information that is more privileged or is to be kept more secure than information in D_i, this restriction is appropriate. However, this security policy can be circumvented in the following way. A process running in D_j could read data in D_j and then copy that data into D_i. Subsequently, a process running in D_i could access the information.

b. An approach to dealing with this problem, known as a trusted system, is discussed in Chapter 10.

4.5 Suppose that the directory **d** and the file **f** have the same owner and group and that **f** contains the text *something*. Disregarding the superuser, no one besides the owner of **f** can change its contents can change its contents, because only the owner has write permission. However, anyone in the owner's group has write permission for **d**, so that any such person can remove **f** from **d** and install a different version, which for most purposes is the equivalent of being able to modify **f**. This example is from Grampp, F., and Morris, R. "UNIX Operating System Security." *AT&T Bell Laboratories Technical Journal*, October 1984.

4.6 A default UNIX file access of full access for the owner combined with no access for group and other means that newly created files and directories will only be accessible by their owner. Any access for other groups or users must be explicitly granted. This is the most common default, widely used by government and business where the assumption is that a person’s work is assumed private and confidential.

A default of full access for the owner combined with read/execute access for group and none for other means newly created files and directories are accessible by all members of the owner’s group. This is suitable when there is a team of people working together on a server, and in general most work is shared with the group.

However there are also other groups on the server for which this does not apply. An organization with cooperating teams may choose this.
A default of full access for the owner combined with read/execute access for both group and other means newly created files and directories are accessible by all users on the server. This is appropriate for organization's where users trust each other in general, and assume that their work is a shared resource. This used to be the default for University staff, and in some research labs. It is also often the default for small businesses where people need to rely on and trust each other.

4.7 In order to provide the Web server access to a user's 'public_html' directory, then search (execute) access must be provided to the user's home directory (and hence to all directories in the path to it), read/execute access to the actual Web directory, and read access to any Web pages in it, for others (since access cannot easily be granted just to the user that runs the web server). However this access also means that any user on the system (not just the web server) has this same access. Since the contents of the user's web directory are being published on the web, local public access is not unreasonable (since they can always access the files via the web server anyway). However in order to maintain these required permissions, if the system default is one of the more restrictive (and more common) options, then the user must set suitable permissions every time a new directory or file is created in the user's web area. Failure to do this means such directories and files are not accessible by the server, and hence cannot be access over the web. This is a common error. As well the fact that at least search access is granted to the user's home directory means that some information can be gained on its contents by other user's, even if it is not readable, by attempting to access specific names. It also means that if the user accidentally grants too much access to a file, it may then be accessible to other users on the system. If the user's files are sufficiently sensitive, then the risk of accidental leakage due to inappropriate permissions being set may be too serious to allow such a user to have their own web pages.

4.8 **a.** $\sum_{i=1}^{N}(U_i \times P_i)$

b. $\sum_{i=1}^{N}(U_i + P_i)$

4.9 For a limited role hierarchy, a role is limited to a single immediate descendent. Enough links must be removed from the diagram to satisfy this requirement.

4.10 a. $r_1 \succ\succ r_2 \Rightarrow$ *authorized_permissions*$(r_2) \subseteq$ *authorized_permissions*$(r_1) \wedge$ *authorized_users*$(r_2) \subseteq$ *authorized_users*(r_1)

b. $r \succ\succ r_1 \wedge r \succ\succ r_2 \Rightarrow r1 = r2$

4.11 a. Role(x) > Role(y) ⇔ Role(x).Position > Role(y).Position ∧ Role(x).Function = Role(y).Function

b. Role(x) > Role(y) ⇔ Role(x).Position > Role(x).Function = Role(y).Function

4.12 RBAC Roles:

Adult premium user role
Adult regular user role
Juvenile premium user role
Juvenile regular user role
Child premium user role
Child regular user role

RBAC Permissions

Can view R rated new release
Can view R rated old release
Can view PG-13 rated new release
Can view PG-13 rated old release
Can view G rated new release
Can view G rated old release

CHAPTER 5 DATABASE AND CLOUD SECURITY

ANSWERS TO QUESTIONS

5.1 A **database** is a structured collection of data stored for use by one or more applications. In addition to data, a database contains the relationships between data items and groups of data items. A **database management system (DBMS)**, which is a suite of programs for constructing and maintaining the database and for offering ad hoc query facilities to multiple users and applications. A **query language** provides a uniform interface to the database for users and applications.

5.2 A relational database is a collection of tables (also called relations). Individual values in a table can be used to link one table to another.

5.3 Whereas the value of a primary key must be unique for each tuple (row) of its table, a foreign key value can appear multiple times in a table.

5.4 **Centralized administration:** A small number of privileged users may grant and revoke access rights.
Ownership-based administration: The owner (creator) of a table may grant and revoke access rights to the table.
Decentralized administration: In addition to granting and revoking access rights to a table, the owner of the table may grant and revoke authorization to other users, allowing them to grant and revoke access rights to the table.

5.5 The grant option enables an access right to cascade through a number of users. If a user has an access right with grant option, the user may pass the right to another user.

5.6 The inference problem arises when the combination of a number of data items is more sensitive than the individual items, or when a combination of data items can be used to infer data of a higher sensitivity.

5.7 **Key management:** Authorized users must have access to the decryption key for the data for which they have access. Because a database is typically accessible to a wide range of users and a number of applications, providing secure keys to selected parts of the database to authorized users and applications is a complex task.
Inflexibility: When part or all of the database is encrypted, it becomes more difficult to perform record searching.

5.8 **Software as a service (SaaS)**: Provides service to customers in the form of software, specifically application software, running on and accessible in the cloud.
Platform as a service (PaaS): Provides service to customers in the form of a platform on which the customer's applications can run.
Infrastructure as a service (IaaS): Provides the customer access to the underlying cloud infrastructure.

5.9 The NIST cloud computing reference architecture focuses on the requirements of "what" cloud services provide, not a "how to" design solution and implementation. The reference architecture is intended to facilitate the understanding of the operational intricacies in cloud computing. It does not represent the system architecture of a specific cloud computing system; instead it is a tool for describing, discussing, and developing a system-specific architecture using a common framework of reference.

5.10 **Abuse and nefarious use of cloud computing**: For many CPs, it is relatively easy to register and begin using cloud services, some even offering free limited trial periods. This enables attackers to get inside the cloud to conduct various attacks, such as spamming, malicious code attacks, and denial of service.
Insecure interfaces and APIs: CPs expose a set of software interfaces or APIs that customers use to manage and interact with cloud services. The security and availability of general cloud services is dependent upon the security of these basic APIs. From authentication and access control to encryption and activity monitoring, these interfaces must be designed to protect against both accidental and malicious attempts to circumvent policy.
Malicious insiders: Under the cloud computing paradigm, an organization relinquishes direct control over many aspects of security and, in doing so, confers an unprecedented level of trust onto the CP. One grave concern is the risk of malicious insider activity. Cloud architectures necessitate certain roles that are extremely high-risk.
Shared technology issues: IaaS vendors deliver their services in a scalable way by sharing infrastructure. Often, the underlying components that make up this infrastructure (CPU caches, GPUs, etc.) were not designed to offer strong isolation properties for a multi-

tenant architecture. CPs typically approach this risk by the use of isolated virtual machines for individual clients. This approach is still vulnerable to attack, by both insiders and outsiders, and so can only be a part of an overall security strategy.

Data loss or leakage: For many clients, the most devastating impact from a security breach is the loss or leakage of data. We address this issue in the next section.

Account or service hijacking: Account and service hijacking, usually with stolen credentials, remains a top threat. With stolen credentials, attackers can often access critical areas of deployed cloud computing services, allowing them to compromise the confidentiality, integrity, and availability of those services.

Unknown risk profile: In using cloud infrastructures, the client necessarily cedes control to the cloud provider on a number of issues that may affect security. Thus the client must pay attention to and clearly define the roles and responsibilities involved for managing risks. For example, employees may deploy applications and data resources at the CP without observing the normal policies and procedures for privacy, security, and oversight.

ANSWERS TO PROBLEMS

5.1

Course Name	Course Number	Day	Time	Room Number	Max Enrollment

Faculty Name	Course 1	Course 2	Course 3

Student Name	Course 1	Course 2	Course 3

5.2 It is clear that Climber-ID is the primary key of the table. The first row cannot be added. If violates the uniqueness property of the key because there is a Climber-ID 214 already in the table. The second row cannot be added. It violates the integrity constraint of the key because there is no value for the primary key. The third row can be added.

5.3 **a.** (1) Updating an owner's name or other data must be done in (potentially) many rows

(2) Possibly incorrect, inconsistent owner data across rows (change in one row, but not in another)
(3) Entering correct data inconsistently
(4) No place to store owner (your customer) data unless they have a pet

b.

PET

PetID	Pet Name	Type	Breed	DOB	Owner Phone

OWNER

Owner Name	Owner Phone	Owner Email

5.4
```
CREATE TABLE student (
        sid INTEGER PRIMARY KEY,
        sname VARCHAR (25),
        sphone CHAR(10)    )
```

5.5 a. This statement retrieves the 'id', 'forename' and 'surname' columns from the 'authors' table, returning all rows in the table that match forename = 'john' and surname = 'smith'.

b. The 'query string' becomes this:

SELECT id, forename, surname FROM authors WHERE forename = 'jo'hn' AND surname = 'smith'

When the database attempts to run this query, it is likely to return an error. The reason is that the insertion of the 'single quote' character 'breaks out' of the single-quote delimited data. The database then tried to execute 'hn' and fails.

c. The authors table will be deleted.

5.6 a. SELECT accounts FROM users WHERE login='doe' AND pass='secret' AND pin=123

b. SELECT accounts FROM users WHERE login='' or 1=1 -- AND pass='' AND pin=
The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them. The returned set evaluates to a nonnull value, which causes the application to conclude that the user authentication was successful.

Therefore, the application would invoke method displayAccounts() and show all of the accounts in the set returned by the database.

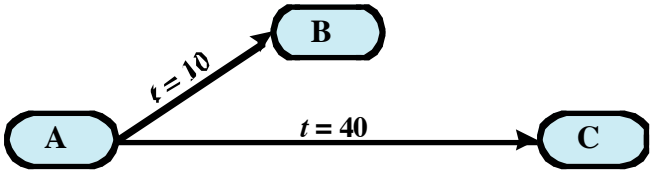
5.7 This produces the following query:

SELECT accounts FROM users WHERE login='' UNION
SELECT cardNo from CreditCards where acctNo=10032 -- AND pass='' AND pin=

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "CreditCards" table. In this case, the database would return column "cardNo" for account "10032." The database takes the results of these two queries, unions them, and returns them to the application. In many applications, the effect of this operation is that the value for "cardNo" is displayed along with the account information.

5.8 The grant of the DELETE privilege by X at time t = 25 must be revoked because X's earliest remaining DELETE privilege was received at time t = 30. But X's grants of READ and INSERT are allowed to remain because they are still supported by incoming grants that occurred earlier in time.

5.9



5.10 a. Consider the the concept of a noncascading revocation, defined as follows. Whenever user A revokes a privilege from user B, authorizations granted by B are not revoked; instead they are respecified as if they had been granted by S, the user issuing revocation. The semantics of the revocation without cascade is to produce the authorization state that would have resulted if the revoker (A) had granted the authorizations that were granted by the revokee (B).

b. The disadvantage of the cascading revocation is that it can be disruptive. In many organizations the authorizations a user possesses are related to his or her particular task or function within the organization. If a user changes task or function (e.g., is promoted), it is desirable to remove only the authorizations of this user, without triggering recursive revocation of all authorizations granted by this user. The disadvantage of the noncascading revocation would be if there are instances when we would like to revoke a user's authorization and simultaneously undo any authorizations by that user. This could occur if the user comes under suspicion.

5.11

User	Permission level
Accounts payable clerk	Should be able to access and change all data.
Installation foreman	Needs to access but not change parts information. Probably does not need to have access to any vendor information except perhaps name.
Receiving clerk	Needs to be able to access and change parts information, such as number in stock. Should be able to access but not change vendor information.

5.12 We assume that there is a unique constraint on flight ID and cargo hold (to prevent scheduling two shipments for the same hold). When a user in role 2 sees that nothing is scheduled for hold C on flight 1254, the user might attempt to insert a new record to transport some vegetables on that flight. However, when he or she attempts to insert the record, the insert will fail due to the unique constraint. At this point, the user has all the data needed to infer that there is a secret shipment on flight 1254. The user could then cross-reference the flight information table to find out the source and destination of the secret shipment and various other information.

5.13
```
GRANT select ON inventory TO hulkhogan, undertaker;
GRANT select ON item TO hulkhogan, undertaker;
```

5.14 Another way is to create a separate (unclassified) relation HIRE-DATES (EMP#, START-DATE). Note that EMP # rather than S# must be used as the key for the new relation, otherwise the inference is not removed.

5.15 If the first two queries are answered, then the **max** query is denied whenever its value is exactly equal to the ratio of the sum and the count values (which happens when all the selected rows have the same salary value). Hence the attacker learns the salary values of all the selected rows when denial occurs.

CHAPTER 6 MALICIOUS SOFTWARE

ANSWERS TO QUESTIONS

6.1 The three broad mechanisms malware can use to propagate are:

- infection of existing executable or interpreted content by viruses that is subsequently spread to other systems;
- exploit of software vulnerabilities either locally or over a network by worms or drive-by-downloads to allow the malware to replicate; and
- social engineering attacks that convince users to bypass security mechanisms to install trojans, or to respond to phishing attacks.

6.2 Four broad categories of payloads that malware may carry are:

- corruption of system or data files;
- theft of service in order to make the system a zombie agent of attack as part of a botnet;
- theft of information from the system, especially of logins, passwords or other personal details by keylogging or spyware programs; and
- stealthing where the malware hides it presence on the system from attempts to detect and block it.

6.3 The characteristics of an advanced persistent threat giving its name are:

- Advanced: Use of a wide variety of intrusion technologies and malware, including the development of custom malware if required.
- Persistent: application of attacks over an extended period against the chosen target in order to maximize the chance of success.
- Threats: a result of the organized, capable, and well-funded attackers intent to compromise the specifically chosen targets.

6.4 The typical phases of operation of a virus or worm are:

- a dormant phase (when the virus is idle),
- a propagation phase (where it makes copies of itself elsewhere),
- a triggering phase (when activated), and
- an execution phase (to perform some target function).

6.5 Some mechanisms a virus can use to conceal itself include: encryption, stealth, polymorphism, metamorphism.

6.6 Machine executable viruses infect executable program files to carry out their work in a manner that is specific to a particular operating system

and, in some cases, specific to a particular hardware platform. Macro viruses infect files with macro or scripting code that is used to support active content in a variety of user document types, and is interpreted by an application.

6.7 A worm may access remote systems to propagate using: an electronic mail or instant messenger facility, file sharing, remote execution capability, remote file access or transfer capability, or a remote login capability.

6.8 A "**drive-by-download**" exploits browser vulnerabilities so that when the user views a web page controlled by the attacker, it contains code that exploits some browser bug to download and install malware on the system without the user's knowledge or consent. It differs from a worm since it does not actively propagate as a worm does, but rather waits for unsuspecting users to visit the malicious web page in order to spread to their systems.

6.9 A **trojan** is an (apparently) useful program or utility containing hidden code that, when invoked, performs some unwanted or harmful function. A Trojan enables malware to propagate as it executes with the privileges of the person running it. Trojans are very common on both computer systems, and increasingly on mobile platforms, though much more on Android than on iOS devices.

6.10 A **logic bomb** is code embedded in the malware that is set to "explode" when certain conditions are met, such as the presence or absence of certain files or devices on the system, a particular day of the week or date, a particular version or configuration of some software, or a particular user running the application. When triggered, the bomb executes some payload carried by the malware.

6.11 A **backdoor** is a secret entry point into a program or system that allows someone who is aware of the backdoor to gain access without going through the usual security access procedures.
A **bot** subverts the computational and network resources of the infected system for use by the attacker.
A **keylogger** captures keystrokes on the infected machine, to allow an attacker to monitor sensitive information including login and password credentials.
Spyware subverts the compromised machine to allow monitoring of a wide range of activity on the system, including monitoring the history and content of browsing activity, redirecting certain web page requests to fake sites controlled by the attacker, dynamically modifying data exchanged between the browser and certain web sites of interest;

which can result in significant compromise of the user's personal information.
A **rootkit** is a set of programs installed on a system to maintain covert access to that system with administrator (or root) privileges, whilst hiding evidence of its presence to the greatest extent possible.
These **can** all be present in the same malware.

6.12 A **phishing** attack uses a spam e-mail to exploit social engineering to leverage user's trust by masquerading as communications from a trusted source, that may direct a user to a fake Web site, or to complete some enclosed form and return in an e-mail accessible to the attacker. A more dangerous variant of this is the **spear-phishing** attack. This again is an e-mail claiming to be from a trusted source. However, the recipients are carefully researched by the attacker, and each e-mail is carefully crafted to suit its recipient specifically, often quoting a range of information to convince them of its authenticity. This greatly increases the likelihood of the recipient responding as desired by the attacker.

6.13 A rootkit may be placed in: user mode where it can intercept calls to APIs and modify results; in kernel mode where it can intercept kernel API calls and hide it presence in kernel tables; in a virtual machine hypervisor where it can then transparently intercept and modify states and events occurring in the virtualized system; or in some other external mode such as BIOS or in BIOS or system management mode, where it can directly access hardware.

6.14 Malware countermeasure elements include:
- **prevention** in not allowing malware to get into the system in the first place, or blocking its ability to modify the system, via policy, awareness, vulnerability mitigation and threat mitigation;
- **detection** to determine that it has occurred and locate the malware;
- **identification** to identify the specific malware that has infected the system; and
- **removal** to remove all traces of malware virus from all infected systems so that it cannot spread further.

6.15 Three places malware mitigation mechanisms may be located, are:
- **on the infected system**, where some host-based "anti-virus" program is running, monitoring data imported into the system, and the execution and behavior of programs running on the system;
- as **part of the perimeter security mechanisms** used in an organizations firewall and intrusion detection systems;
- or it may use **distributed mechanisms** that gather data from both host-based and perimeter sensors, potentially over a large number

of networks and organizations, in order to obtain the largest scale view of the movement of malware.

6.16 The four generations of anti-virus software are:

- **First generation**: simple scanners that require a malware signature to identify it
- **Second generation**: heuristic scanners use heuristic rules to search for probable malware instances, or uses integrity checking to identify changed files
- **Third generation**: activity traps that identify malware by its actions rather than its structure in an infected program
- **Fourth generation**: full-featured protection uses packages of a variety of anti-virus techniques used in conjunction, including scanning and activity trap components.

ANSWERS TO PROBLEMS

6.1 The program will loop indefinitely once all of the executable files in the system are infected.

6.2 D is supposed to examine a program P and return TRUE if P is a computer virus and FALSE if it is not. But CV calls D. If D says that CV is a virus, then CV will not infect an executable. But if D says that CV is not a virus, it infects an executable. D always returns the wrong answer.

6.3 The original code has been altered to disrupt the signature without affecting the semantics of the code. The ineffective instructions in the metamorphic code are the second, third, fifth, sixth, and eighth.

6.4 **a.** The following is from Spafford, E. " The Internet Worm Program: An Analysis." Purdue Technical Report CSD-TR-823.
Common choices for passwords usually include fantasy characters, but this list contains none of the likely choices (e.g., "hobbit", "dwarf", "gandalf", "skywalker", "conan"). Names of relatives and friends are often used, and we see women's names like "jessica", "caroline", and "edwina", but no instance of the common names "jennifer" or "kathy". Further, there are almost no men's names such as "thomas" or either of "stephen" or "steven" (or "eugene"!). Additionally, none of these have the initial letters capitalized, although that is often how they are used in passwords. Also of interest, there are no obscene words in this dictionary, yet many reports of concerted password cracking experiments have revealed that there are a significant number of users who use such words (or phrases) as passwords. The list contains at least one incorrect spelling: "commrades" instead of "comrades"; I also believe that "markus" is a misspelling of "marcus". Some of the words

do not appear in standard dictionaries and are non-English names: "jixian", "vasant", "puneet", etc. There are also some unusual words in this list that I would not expect to be considered common: "anthropogenic", "imbroglio", "umesh", "rochester", "fungible", "cerulean", etc.

b. Again, from Spafford:
I imagine that this list was derived from some data gathering with a limited set of passwords, probably in some known (to the author) computing environment. That is, some dictionary-based or brute-force attack was used to crack a selection of a few hundred passwords taken from a small set of machines. Other approaches to gathering passwords could also have been used: Ethernet monitors, Trojan Horse login programs, etc. However they may have been cracked, the ones that were broken would then have been added to this dictionary. Interestingly enough, many of these words are not in the standard on-line dictionary (in /usr/dict/words). As such, these words are useful as a supplement to the main dictionary-based attack the worm used as strategy #4, but I would suspect them to be of limited use before that time.

6.5 Logic bomb.

6.6 Backdoor.

6.7 The found USB memory stick may pose a range of threats to the confidentiality, integrity and availability of the work system. Each of the malware propagation mechanisms we discuss could use such a memory stick for transport. It may carry a program infected with an executable virus, or document infected with a macro virus, which if run or opened can allow the virus to run and spread. It could carry a malicious worm that may be run automatically using the autorun capability, or by exploiting some vulnerability when the USB stick is viewed. Or it could contain a trojan horse program or file that would threaten the system if installed or allowed to run. You can mitigate these threats, and try to safely determine the contents of the memory stick, by scanning the memory stick with suitable, up-to-date anti-virus software for any signs of malware – though this will not detect unknown, zero-day exploits. You could examine the memory stick in a controlled environment, such as a live-boot linux or other system, or in some emulation environment, which cannot be changed even if the malware does manage to run.

6.8 Observations of your home PC is responding very slowly, with high levels of network activity, may indicate the presence of malware, likely including bot code, on the system. The slow response and net traffic could be caused by it participating in a botnet, perhaps distributing spam emails, performing DDoS attacks, or other malicious activities.

This malware could have gained access to the system as a result of installing some trojan program perhaps advertised in spam email or on a compromised website, from a drive-by-download, or from exploit of some vulnerability on the system by a worm. Possible steps to check whether this has occurred include examining the process/task list for unknown programs executing, looking at logs of network traffic kept by a host firewall program to see which programs are generating traffic, or scanning the system with suitable, up-to-date anti-virus software for any signs of malware – though this will not detect unknown, zero-day exploits. If you do identify malware on your PC, you may be able to restore it to safe operation using suitable, up-to-date anti-virus software, provided the malware is known. Otherwise you may have to erase all storage and rebuild the system from scratch.

6.9 If a user installs some custom codec claimed needed to view some videos, they may actually be installing trojan horse code. It may indeed allow viewing of the video, or may just be an excuse to compromise the system. Such code may pose a range of threats to the confidentiality, integrity and availability of the system. It may include backdoor, bot, keylogger, spyware, rootkit or indeed any other malware payloads.

6.10 If when you download and start to install some game app, you are asked to approve the access permissions "Send SMS messages" and to "Access your address-book", you should indeed be suspicious that a game wants these types of permissions, as it would not seem needed just for a game. Rather it could be malware that wants to collect details of all your contacts, and either return them to the attacker via SMS, or allow the code to send SMS messages to your contacts, perhaps enticing them to also download and install this malware. Such code is a trojan horse, since it contains covert functions as well as the advertised functionality.

6.11 If you should open the PDF attachment, then it could contain malicious scripting code that could run should you indeed select the 'Open' button. This may be either worm (specifically exploiting a client-side vulnerability), or trojan horse code. You could you check your suspicions without threatening your system by using the scroll bar to examine all the code about to be executed should you select the 'Open' button, and see if it looks suspicious. You could also scan the PDF document with suitable, up-to-date anti-virus software for any signs of malware – though this will not detect unknown, zero-day exploits. This type of message is associated with a spear-phishing attack, given that the email was clearly crafted to suit the recipient. That particular e-mail would only have been sent to one or a few people for whom the details would seem plausible.

6.12 This email is attempting a general phishing attack, being sent to very large numbers of people, in the hope that a sufficient number both use the named bank, and are fooled into divulging their sensitive login credentials to the attacker. The most likely mechanism used to distribute this e-mail is via a botnet using large numbers of compromised systems to generate the necessary high volumes of spam emails. You should never ever follow such a link in an email and supply the requested details. You should only ever access sensitive sites by directly entering their known URL into your browser. It may be appropriate to forward a copy of such emails to a relevant contact at the bank if they ask for this. Otherwise it should just be deleted.

6.13 Such a letter strongly suggests that an attacker has collected sufficient personal details about you in order to satisfy the finance company that they are you for the purpose of establishing such a loan. Having taken the money, they have then left you responsible for the repayments. This was most likely done using either a phishing attack, perhaps persuading you to complete and return some form with the needed personal details; or by using spyware installed on your personal computer system by a worm or trojan horse malware, that then collected the necessary details from files on the system, or by monitoring your access to sensitive sites, such as banking sites.

6.14 A (host-based) personal firewall monitors and controls network traffic flowing between the host and the net, and can help protect against exploit of remotely accessible network server vulnerabilities by blocking access to all but required services. Anti-virus software helps protect against the import and use of malware however it enters the system. Anti-virus software helps block the spread of macro viruses spread using email attachments. A (host-based) personal firewall could block the use of backdoors on the system.

CHAPTER 7 DENIAL-OF-SERVICE ATTACKS

ANSWERS TO QUESTIONS

7.1 A **denial of service** (DoS) attack is an action that prevents or impairs the authorized use of networks, systems, or applications by exhausting resources such as central processing units (CPU), memory, bandwidth, and disk space.

7.2 Resources that could be attacked include any limited resources such as: network bandwidth, system resources, or application resources.

7.3 The goal of a flooding attack is generally to overload the network capacity on some link to a server, or alternatively to overload the server's ability to handle and respond to this traffic.

7.4 Virtually any type of network packet can be used in a flooding attack, though common flooding attacks use ICMP, UDP or TCP SYN packet types.

7.5 Many DoS attacks use packets with spoofed source addresses so any responses packets that result are no longer be reflected back to the original source system, but rather are scattered across the Internet to all the various forged source addresses. Some of these addresses might correspond to real systems, others may not be used, or not reachable. Any response packets returned as a result only add to the flood of traffic directed at the target system.

7.6 "**backscatter traffic**" are packets generated in response to a DoS attack packet with a forged random source address, e.g. the ICMP echo response from an ICMP echo request being used to flood a link. Monitoring these packets, which are randomly distributed over the Internet, gives valuable information on the type and scale of attacks. This **backscatter traffic** provides information on any DoS attacks that use a forged random source address with the destination address being the target, including various single and distributed flooding attacks, and syn spoofing attacks. It does not provide information on attacks that do

not use randomly forged source addresses, or reflection or amplification attacks where the forged source address is that of the desired target.

7.7 A distributed denial of service (DDoS) attack uses multiple attacking systems, often using compromised user workstations or PC's. Large collections of such systems under the control of one attacker can be created, collectively forming a "botnet". By using multiple systems, the attacker can significantly scale up the volume of traffic that can be generated. Also by directing the attack through intermediaries, the attacker is further distanced from the target, and significantly harder to locate and identify.

7.8 Distributed denial of service (DDoS) attack botnets typically use a control hierarchy, where a small number of systems act as handlers controlling a much larger number of agent systems, as shown in Figure 7.4. These have are a number of advantages, as the attacker can send a single command to a handler, which then automatically forwards it to all the agents under its control. Automated infection tools can also be used to scan for and compromise suitable zombie systems.

7.9 In a reflection attack, the attacker sends a network packet with a spoofed source address to a service running on some network server, that responds to the spoofed source address that belongs to the actual attack target. If the attacker sends a number of such spoofed requests to a number of servers, the resulting flood of responses can overwhelm the target's network link. The fact that normal server systems are being used as intermediaries, and that their handling of the packets is entirely conventional, means these attacks can be easier to deploy, and harder to trace back to the actual attacker.

7.10 An amplification attack involves sending packets to intermediaries with a spoofed source address for the target system. They differ in generating multiple response packets for each original packet sent, typically by directing the original request to the broadcast address for some network. Alternatively they use a service, often DNS, that can generate a much larger response packet than the original request.

7.11 The primary defense against many DoS attacks is to prevent source address spoofing. This must be implemented close to the source of any packet, when the real address (or at least network) is known. Typically this is the ISP providing the network connection for an organization or home user. It knows which addresses are allocated to all its customers, and hence is best placed to ensure that valid source addresses are used in all packets from its customers.

7.12 Non-spoofed flooding attacks are best defended against by the provision of significant excess network bandwidth and replicated distributed servers, particularly when the overload is anticipated. This does have a significant implementation cost though. Rate limits of various types on traffic can also be imposed. However such attacks cannot be entirely prevented, and may occur "accidentally" as a result of very high legitimate traffic loads.

7.13 It is possible to specifically defend against the SYN spoofing attack by using a modified version of the TCP connection handling code, which instead of saving the connection details on the server, encodes critical information in a "cookie" sent as the server's initial sequence number. When a legitimate client responds with an ACK packet, the server is able to reconstruct this information. Typically this technique is only used when the table overflows, as it does take computation resources on the server, and also blocks the use of certain TCP extensions.

7.14 Like all the reflection-based attacks, the basic defense against DNS amplification attacks is to prevent the use of spoofed source addresses. This filtering needs to be done as close to the source as possible, by routers or gateways knowing the valid address ranges of incoming packets. Typically this is the ISP providing the network connection for an organization or home user. Otherwise, appropriate configuration of DNS servers, in particular limiting recursive responses to internal client systems only, as described in RFC 5358, can restrict some variants of DNS amplification attacks.

7.15 To prevent an organization's systems being used as intermediaries in a broadcast amplification attack, the best defense is to block the use of IP directed broadcasts. This can be done either by the ISP, or by any organization whose systems could potentially be used as an intermediary.

7.16 The terms **slashdotted** or **flash crowd** refer to very large volumes of legitimate traffic, as result of high publicity about a specific site, often as a result of a posting to the well-known Slashdot or other similar news aggregation site. There is very little that can be done to prevent this type of either accidental or deliberate overload, without also compromising network performance. The provision of significant excess network bandwidth and replicated distributed servers is the usual response as noted in question 7.12.

7.17 In order to successfully respond to a denial of service attack, a good incident response plan is needed to provide guidance. When a denial of service attack is detected, the first step is to identify the type of attack and hence the best approach to defend against it. From this analysis

the type of attack is identified, and suitable filters designed to block the flow of attack packets. These have to be installed by the ISP on their routers. If the attack targets a bug on a system or application, rather than high traffic volumes, then this must be identified, and steps taken to correct it to prevent future attacks. In the case of an extended, concerted, flooding attack from a large number of distributed or reflected systems, it may not be possible to successfully filter enough of the attack packets to restore network connectivity. In such cases the organization needs a contingency strategy to switch to alternate backup servers, or to rapidly commission new servers at a new site with new addresses, in order to restore service.

7.18 The organization may wish to trace the source of various types of packets used in a DoS attack. If non-spoofed addresses are used, this is easy. However if spoofed sources addresses are used, this can be difficult and time-consuming, as their ISP will need to trace the flow of packets back in an attempt to identify their source. This is generally neither easy nor automated, and requires cooperation from the network providers these packets traverse.

ANSWERS TO PROBLEMS

7.1 In a DoS attack using ICMP Echo Request (ping) packets 500 bytes in size, to flood a target organization using a 0.5 Megabit per second (Mbps) link the attacker needs 500000 / (500 × 8) = 125 packets per second. On a 2-Mbps link its 2000000 / (500 × 8) = 500 packets per second. On a 10-Mbps link its 10000000 / (500 × 8) = 2500 packets per second.

7.2 For a TCP SYN spoofing attack, on a system with a table for 256 connection requests, that will retry 5 times at 30 second intervals, before purging the request from its table, each connection request occupies a table entry for 6 × 30 secs (initial + 5 repeats) = 3 min. In order to ensure that the table remains full, the attacker must continue to send 256/ 3 or about 86 TCP connection requests per minute? Assuming the TCP SYN packet is 40 bytes in size, this consumes about 86 × 40 × 8 / 60, which is about 459 bits per second, a negligible amount.

7.3 In the distributed variant of the attack from Problem 7.1, a single zombie PC can send 128000 / (500 × 8) = 32 packets per second. About 4 such zombie systems are needed to flood a target organization using a 0.5 Megabit per second (Mbps) link, looking either at 500kbps / 128 kbps, or 125 / 32 packets per sec. For a 2Mbps link about 16 are needed (500/32 pps), for a 10-Mbps link about 79 are needed (2500/32

pps). Given reports of botnets composed of many thousands of zombie systems, clearly multiple such simultaneously DDoS attacks are possible. As is an attack on a major organization with multiple, much larger network links (e.g. 1000 zombies with 128-kbps links can flood 128 Mbps of network link capacity).

7.4 The answers for the DNS amplification attack are the same as in Problem 7.1. On a 0.5-Mbps link, 125 packets, each of 500 bytes, are needed per second. 500 pps are needed to flood a 2-Mbps link, and 2500 pps to flood a 10-Mbps link. Assuming a 60-byte DNS request packet then $125 \times 60 \times 8 = 60$ kbps is needed to trigger the flood on a 0.5-Mbps link, 240 kbps to flood the 2-Mbps link, and 1.2 Mbps to flood the 10-Mbps link. In all cases the amplification is 500 / 60 = 8.3 times.

7.5 The answer to this question depends on the operating system (and version) chosen, however SYN Cookies are now supported on many systems.

7.6 The answer to this question also depends on the type of router investigated, but again these features are common on enterprise grade devices.

7.7 In this future idealized more secure network, administrators of server systems still do need to be concerned about, and take further counter-measures against, DoS attacks. Attacks using real addresses from real systems with high bandwidth network connections are still possible, as are "flash-crowd" overloads as a result of, possible fraudulent, publicity. To reduce the impact of such attacks, measures are needed to manage intermittent high traffic volumes, as mentioned in review questions 7.11 and 7.13.

7.8 The results of practical lab experiments such as these depend on the facilities and equipment available.

CHAPTER 8 INTRUSION DETECTION

ANSWERS TO QUESTIONS

8.1 Four classes of intruders we discuss are:

- **Cyber criminals** are either individuals or members of an organized crime group with a goal of financial reward;
- **Activists**: are either individuals, usually working as insiders, or members of a larger group of outsider attackers, who are motivated by social or political causes;
- **State-sponsored organizations**: are groups of hackers sponsored by governments to conduct espionage or sabotage activities. They are also known as Advanced Persistent Threats (APTs), due to the covert nature and persistence over extended periods involved with many attacks in this class; and
- **Others**: are hackers with motivations other than those listed above, including classic hackers or crackers who are motivated by technical challenge or by peer-group esteem and reputation.

8.2 The steps typically used by intruders when attacking a system are:

- **Target Acquisition and Information Gathering**: where the attacker identifies and characterizes the target systems using publicly available information, both technical and non-technical, and the use network exploration tools to map target resources;
- **Initial Access**: typically by exploiting a remote network vulnerability, by guessing weak authentication credentials used in a remote service, or via the installation of malware on the system using some form of social engineering or drive-by-download attack;
- **Privilege Escalation**: are actions taken on the system, typically via a local access vulnerability to increase the privileges available to the attacker to enable their desired goals on the target system;
- **Information Gathering or System Exploit**: are actions by the attacker to access or modify information or resources on the system, or to navigate to another target system;
- **Maintaining Access**: are actions such as the installation of backdoors or other malicious software, or the addition of covert authentication credentials or other configuration changes to the system, to enable continued access by the attacker after the initial attack.

- **Covering Tracks**: where the attacker disables or edits audit logs to remove evidence of attack activity, and uses rootkits and other measures to hide covertly installed files or code

8.3 Table 8.1 lists examples of activities associated with each of the attack steps that may be used by an intruder. Pick any item from each part.

8.4 **Sensors:** Sensors are responsible for collecting data. The input for a sensor may be any part of a system that could contain evidence of an intrusion. Types of input to a sensor include network packets, log files, and system call traces. Sensors collect and forward this information to the analyzer.
Analyzers: Analyzers receive input from one or more sensors or from other analyzers. The analyzer is responsible for determining if an intrusion has occurred. The output of this component is an indication that an intrusion has occurred. The output may include evidence supporting the conclusion that an intrusion occurred. The analyzer may provide guidance about what actions to take as a result of the intrusion.
User interface: The user interface to an IDS enables a user to view output from the system or control the behavior of the system. In some systems, the user interface may equate to a manager, director, or console component.

8.5 **Host-based IDS:** Monitors the characteristics of a single host and the events occurring within that host for suspicious activity
Network-based IDS: Monitors network traffic for particular network segments or devices and analyzes network, transport, and application protocols to identify suspicious activity
Distributed or hybrid IDS: Combines information from a number of sensors, often both host and network-based, in a central analyzer that is able to better identify and respond to intrusion activity.

8.6 **1.** If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.
2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

8.7 A **false positive**, or false alarm, is where authorized users are identified as intruders by an IDS. A **false negative** is when intruders

are not identified as intruders by an IDS, as a result of a tighter interpretation of intruder behavior in an attempt to limit false positives.

8.8 The base-rate fallacy occurs when there is an attempt to detect a phenomenon that occurs rarely. The frequency of occurrence is referred to as the base rate. When the base rate is very low, it is difficult to achieve low levels of both false positives and false negatives.

8.9 •Run continually with minimal human supervision.
•Be fault tolerant in the sense that it must be able to recover from system crashes and reinitializations.
•Resist subversion. The IDS must be able to monitor itself and detect if it has been modified by an attacker.
•Impose a minimal overhead on the system where it is running.
•Be able to be configured according to the security policies of the system that is being monitored.
•Be able to adapt to changes in system and user behavior over time.
•Be able to scale to monitor a large number of hosts.
•Provide graceful degradation of service in the sense that if some components of the IDS stop working for any reason, the rest of them should be affected as little as possible.
•Allow dynamic reconfiguration; that is, the ability to reconfigure the IDS without having to restart it.

8.10 **Anomaly detection** involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.
Signature or Heuristic detection uses a set of known malicious data patterns (signatures) or attack rules (heuristics) that are compared with current behavior to decide if is that of an intruder. It is also known as misuse detection. This approach can only identify known attacks for which it has patterns or rules.

8.11 The three broad categories of classification approaches used by anomaly detection systems are:

- **Statistical**: Analysis of the observed behavior using univariate, multivariate, or time-series models of observed metrics.
- **Knowledge based**: Approaches use an expert system that classifies observed behavior according to a set of rules that model legitimate behavior.
- **Machine-learning**: Approaches automatically determine a suitable classification model from the training data using data mining techniques.

8.12 A number of machine-learning approaches used in anomaly detection systems include:

- **Bayesian networks**: Encode probabilistic relationships among observed metrics.
- **Markov models**: Develop a model with sets of states, some possibly hidden, interconnected by transition probabilities.
- **Neural networks**: Simulate human brain operation with neurons and synapse between them, that classify observed data.
- **Fuzzy logic**: Uses fuzzy set theory where reasoning is approximate, and can accommodate uncertainty.
- **Genetic algorithms**: Uses techniques inspired by evolutionary biology, including inheritance, mutation, selection and recombination, to develop classification rules.
- **Clustering and outlier detection**: Group the observed data into clusters based on some similarity or distance measure, and then identify subsequent data as either belonging to a cluster or as an outlier.

8.13 **Signature approaches** match a large collection of known patterns of malicious data against data stored on a system or in transit over a network. **Rule-based heuristic identification** involves the use of rules for identifying known penetrations or penetrations that would exploit known weaknesses. Rules can also be defined that identify suspicious behavior, even when the behavior is within the bounds of established patterns of usage.

8.14 Some data sources used in a HIDS are:

- **System call traces**: A record of the sequence of systems calls by processes on a system;
- **Audit (log file) records**: Most modern operating systems include accounting software that collects information on user activity;
- **File integrity checksums**: Periodically scan critical files for changes from the desired baseline by comparing a current cryptographic checksums for these files, with a record of known good values;
- **Registry access**: An approach used on Windows systems is to monitor access to the registry, given the amount of information and access to it used by pro- grams on these systems.

8.15 Signature and heuristic HIDS are currently more commonly deployed, particularly on Windows systems, due to the difficulty in gathering suitable data to use in anomaly HIDS, and because of the load placed on the system to gather and classify this data. Signature or heuristic based HIDS are widely used, particularly as seen in anti-virus (A/V), more correctly viewed as anti-malware, products. These are very commonly used on Windows systems, and also incorporated into mail

and web application proxies on firewalls and in network based IDSs. These products are quite efficient at detecting known malware, however they are not capable of detecting zero-day attacks that do not correspond to the known signatures or heuristic rules.

8.16 A Distributed HIDS provide a more effective defense by coordination and cooperation among HIDSs across the network.

8.17 An **inline sensor** is inserted into a network segment so that the traffic that it is monitoring must pass through the sensor. A **passive sensor** monitors a copy of network traffic; the actual traffic does not pass through the device.

8.18 Possible locations for NIDs sensors are:

- just inside the external firewall;
- between the external firewall and the Internet or WAN;
- at the entrance to major backbone networks;
- on workstation LANs.

8.19 As with host-based intrusion detection, network-based intrusion detection makes use of signature detection and anomaly detection. Unlike the case with HIDS, a number of commercial anomaly NIDS products are available, as well as more traditional signature detection systems.

8.20 A distributed or hybrid IDS combines in a central IDS, the complementary information sources used by HIDS with host-based process and data details, and NIDS with network events and data, to manage and coordinate intrusion detection and response in an organization's IT infrastructure.

8.21 Honeypots are decoy systems that are designed to lure a potential attacker away from critical systems.

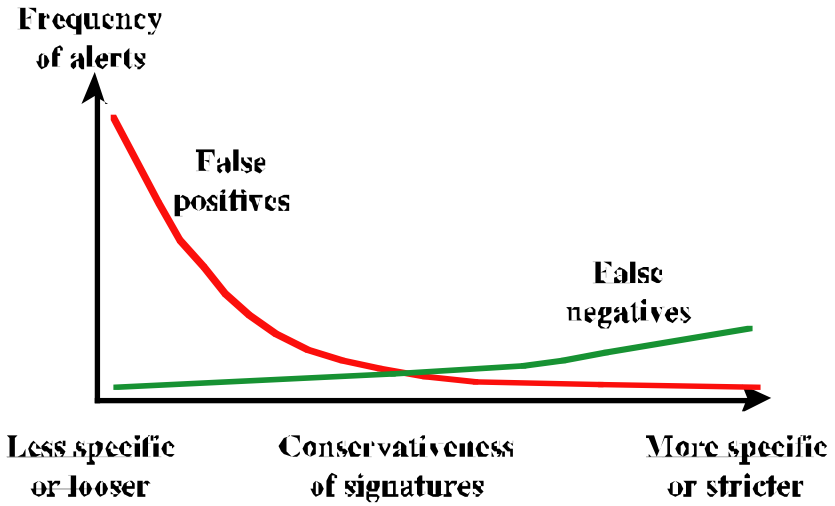
8.22 Honeypots are typically classified as being either a:

- **Low interaction honeypot**: a software package that emulates particular IT services or systems well enough to provide a realistic initial interaction, but does not execute a full version of those services or systems.
- **High interaction honeypot**: a real system, with a full operating system, services and applications, which are instrumented and deployed where they can be accessed by attackers.

ANSWERS TO PROBLEMS

8.1 Types of publicly available information that could be used by an attacker include: information required by law such as business registration and contact details or share registration details; contact information in phone books, DNS entries, network registration and WHOIS details; publicity and contact details provided by an organization on their website, or in publications handed out to the public. This suggests that from a security perspective, the content and detail of such information should be minimized. But this may well conflict with the organization's business and legal requirements to make this information available? It can be very difficult to reconcile these conflicting demands, though the appropriate balance may be suggested by the results of a risk assessment of the organization which may identify which types of information may be particularly dangerous. It may be possible to remove details of individual's names and positions, which would be of use in a spear-phishing attack, and use generic position details instead.

8.2 This is a typical example:



8.3 The following is an extract from [SCAR12]:
Choosing sensor locations for a wireless IDP deployment is a fundamentally different problem than choosing locations for any other type of IDP sensor. If the organization uses WLANs, wireless sensors should be deployed so that they monitor the RF range of the organization's WLANs (both APs and STAs), which often includes mobile components such as laptops and PDAs. Many organizations also want to deploy sensors to monitor physical regions of their facilities where there should be no WLAN activity, as well as channels and bands that the organization's WLANs should not use, as a way of detecting rogue APs and ad hoc WLANs. Other considerations for selecting wireless sensor locations include the following:
Physical Security. Sensors are often deployed into open locations (e.g., hallway ceilings, conference rooms) because their range is much greater there than in closed locations (e.g., wiring closets). Sensors are

sometimes deployed outdoors as well.28 Generally, sensors in open interior locations and external locations are more susceptible to physical threats than other sensors. If the physical threats are significant, organizations might need to select sensors with anti-tamper features or deploy sensors where they are less likely to be physically accessed (e.g., within view of a security camera).

Sensor Range. The actual range of a sensor varies based on the surrounding facilities (e.g., walls, doors). Some wireless IDP vendors offer modeling software that can analyze building floor plans and the attenuation characteristics of walls, doors, and other facility components to determine effective locations for sensors. Sensor range can also vary based on the location of people within the facility and other changing characteristics, so sensors should be deployed so that their ranges have some overlap (e.g., at least 20%).

Wired Network Connections. The sensors typically need to be connected to the wired network. If there is a need to deploy sensors in an area where there is no wired network, then it might be necessary to extend the wired network into that area. This is generally a concern only if the organization wants to monitor portions of their facilities that are outside the range of the WLAN.

Cost. Ideally, an organization could deploy sensors throughout its facilities to perform full wireless monitoring. However, the number of sensors needed to do so can be quite large, especially in wide-open campus environments. Organizations should compare WLAN threats to the cost of sensor purchases, deployment, and maintenance, and develop a solution that creates an acceptable level of risk. For example, an organization might decide to deploy fixed sensors throughout the range of the organization's WLANs, and to do periodic checks of other areas using mobile sensors.

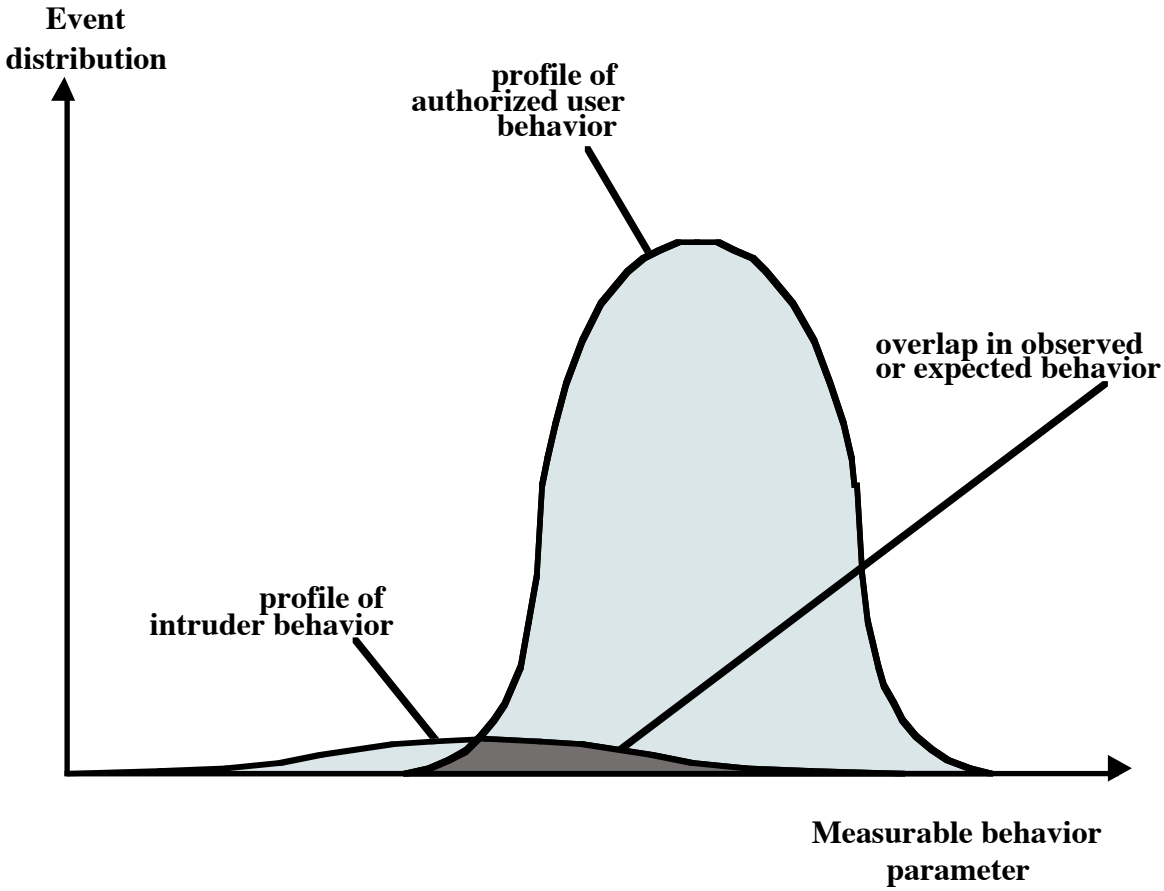
AP and Wireless Switch Locations. If a bundled solution (e.g., wireless IDP software on an AP) would meet the organization's other requirements, then the locations of APs and wireless switches are particularly important because the wireless IDP software could potentially be deployed onto those devices.

8.4 a. This rule wants to catch attempts to create a new database instance. Line 1, the rule header, states that interesting packets are flowing from external IP addresses for database servers responding on Oracle ports. Line 2 is the text alert to be reported. Line 3 defines two additional matching conditions: first, packets must be directed to a server and must be part of an already established TCP connection, and second, the case-independent string "create database" must be contained in the packet payload.

b. Typically, a system administrator would configure a system to forbid database creation from across the Internet. Such attempts would be blocked by the firewall. The external NIDS would simply be a way of

segregating out such attacks and alerting on them. If the NIDS is inside the firewall, it would be able to catch a serious deficiency in firewall behavior.

8.5 a. The graph below doesn't look like a correct probability distribution and is instead labeled as *event distribution*. The point here is that even if you have nice, mostly non-overlapping probability distributions for distinguishing intruders and authorized users like Figure 8.1, the problem is for most systems we hope the actual numbers of intruders is dwarfed by the number of authorized users. This means that the long tail of the authorized users distribution that overlaps with the intruders distribution would generate lots of false positives (relative to the number of real intruders detected) even if it is only a few percent of the authorized users.



b. A randomly selected event that in the overlap region is (roughly) 95% likely to be an authorized user, even though the region covers 50% of the intruders probability distribution.

8.6 A file integrity checking tool such as tripwire can be very useful in identifying changed files or directories on a system, particularly when those change should not have occurred. However most computer systems are not static, and significant numbers of files do change constantly. Hence it is necessary to configure tripwire with a list of files

and directories to monitor, since otherwise reports to the administrator would be filled with lists of files that are changing as a matter of normal operation of the system. It is not too difficult to monitor a small list of critical system programs, daemons and configuration files. Doing this means attempts to alter these files will likely be detected. However the large areas of the system not being monitored means an attacker changing or adding files in these areas will not be detected. The more of the system that is to be monitored, the more care is needed to identify only files not expected to change. Even then, it is likely that user's home areas, and other shared document areas, cannot be monitored, since they are likely to be creating and changing files in there regularly. As well, there needs to be a process to manage the update of monitored files (as a result of installing patches, upgrades, new services, configuration changes etc). This process has to verify that the changed files are correct, and then update the cryptographic checksums of these files. Lastly the database of cryptographic checksums must be protected from any attempt by an attacker to corrupt it, ideally by locating on read-only media (except when controlled updates are occurring).

8.7 This is a conditional probability problem. Total possible combinations for threat level Medium are: (P3, P4), (P4, P3) out of a total number of combinations = 16 – [Number of Events with neither of the two nodes generating a P3 signature] = 16 – 9 = 7 All Possibilities:

(P1, P1)(P1, P2), (P1, P3)(P1, P4)(P2, P1)(P2, P2)(P2, P3)(P2, P4)(P3, P1)(P3, P2)(P3, P3)(P3, P4)(P4, P1)(P4, P2)(P4, P3) (P4, P4)

Therefore, the Probability is = 2/7

Or Let A = {1 P3 and 1 P4}, B = {at least one is P3}
Pr[A|B] = Pr[AB]/ P[B]
Pr[AB] is the probability that one outcome is P3 and one outcome is P4 AND that at least one outcome is P3, is 2/16, and the probability of getting at least one P3 is 7/16, therefore, the

Pr[A|B] = [2/16]/[7/16] = 2/7

8.8 Let WB equal the event {witness reports Blue cab}. Then:

$$\Pr[\text{Blue}/\text{WB}] = \frac{\Pr[\text{WB}/\text{Blue}]\Pr[\text{Blue}]}{\Pr[\text{WB}/\text{Blue}]\Pr[\text{Blue}] + \Pr[\text{WB}/\text{Green}]\Pr[\text{Green}]}$$

$$= \frac{(0.8)(0.15)}{(0.8)(0.15) + (0.2)(0.85)} = 0.41$$

This example, or something similar, is referred to as "the juror's fallacy."

CHAPTER 9 FIREWALLS AND INTRUSION PREVENTION SYSTEMS

ANSWERS TO QUESTIONS

9.1 **1.** All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible, as explained later in this section.
2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies, as explained later in this section.
3. The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system.

9.2 **IP Address and Protocol Values**: Controls access based on the source or destination addresses and port numbers, direction of flow being inbound or outbound, and other network and transport layer characteristics.
Application Protocol: Controls access on the basis of authorized application protocol data.
User Identity: Controls access based on the users identity, typically for inside users who identify themselves using some form of secure authentication technology, such as IPSec.
Network Activity: Controls access based on considerations such as the time or request, or other activity patterns.

9.3 **Source IP address:** The IP address of the system that originated the IP packet.
Destination IP address: The IP address of the system the IP packet is trying to reach.
Source and destination transport-level address: The transport level (e.g., TCP or UDP) port number, which defines applications such as SNMP or TELNET.
IP protocol field: Defines the transport protocol.
Interface: For a router with three or more ports, which interface of the router the packet came from or which interface of the router the packet is destined for.

9.4 **1.** Because packet filter firewalls do not examine upper-layer data, they cannot prevent attacks that employ application-specific vulnerabilities or functions. For example, a packet filter firewall cannot block specific application commands; if a packet filter firewall allows a given application, all functions available within that application will be permitted.
2. Because of the limited information available to the firewall, the logging functionality present in packet filter firewalls is limited. Packet filter logs normally contain the same information used to make access control decisions (source address, destination address, and traffic type).
3. Most packet filter firewalls do not support advanced user authentication schemes. Once again, this limitation is mostly due to the lack of upper-layer functionality by the firewall.
4. They are generally vulnerable to attacks and exploits that take advantage of problems within the TCP/IP specification and protocol stack, such as *network layer address spoofing*. Many packet filter firewalls cannot detect a network packet in which the OSI Layer 3 addressing information has been altered. Spoofing attacks are generally employed by intruders to bypass the security controls implemented in a firewall platform.
5. Finally, due to the small number of variables used in access control decisions, packet filter firewalls are susceptible to security breaches caused by improper configurations. In other words, it is easy to accidentally configure a packet filter firewall to allow traffic types, sources, and destinations that should be denied based on an organization's information security policy.

9.5 A **traditional packet filter** makes filtering decisions on an individual packet basis and does not take into consideration any higher layer context. A **stateful inspection packet filter** tightens up the rules for TCP traffic by creating a directory of outbound TCP connections, as shown in Table 9.2. There is an entry for each currently established connection. The packet filter will now allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory

9.6 An application-level gateway, also called a proxy server, acts as a relay of application-level traffic.

9.7 A circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

9.8 **Packet filtering firewall:** Applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet.

Stateful inspection firewall: Tightens up the rules for TCP traffic by creating a directory of outbound TCP connections, as shown in Table 9.2. There is an entry for each currently established connection. The packet filter will now allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory.

Application proxy firewall: Acts as a relay of application-level traffic (Figure 9.1d). The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of an application that the network administrator considers acceptable while denying all other features

Circuit-level proxy firewall: As with an application gateway, a circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

9.9 • The bastion host hardware platform executes a secure version of its operating system, making it a hardened system.

• Only the services that the network administrator considers essential are installed on the bastion host. These could include proxy applications for DNS, FTP, HTTP, and SMTP.

• The bastion host may require additional authentication before a user is allowed access to the proxy services. In addition, each proxy service may require its own authentication before granting user access.

• Each proxy is configured to support only a subset of the standard application's command set.

• Each proxy is configured to allow access only to specific host systems. This means that the limited command/feature set may be applied only to a subset of systems on the protected network.

• Each proxy maintains detailed audit information by logging all traffic, each connection, and the duration of each connection. The audit log is an essential tool for discovering and terminating intruder attacks.

• Each proxy module is a very small software package specifically designed for network security. Because of its relative simplicity, it is easier to check such modules for security flaws. For example, a typical UNIX mail application may contain over 20,000 lines of code, while a mail proxy may contain fewer than 1000.
• Each proxy is independent of other proxies on the bastion host. If there is a problem with the operation of any proxy, or if a future vulnerability is discovered, it can be uninstalled without affecting the operation of the other proxy applications. Also, if the user population requires support for a new service, the network administrator can easily install the required proxy on the bastion host.
• A proxy generally performs no disk access other than to read its initial configuration file. Hence, the portions of the file system containing executable code can be made read only. This makes it difficult for an intruder to install Trojan horse sniffers or other dangerous files on the bastion host.
• Each proxy runs as a nonprivileged user in a private and secured directory on the bastion host.

9.10 • Filtering rules can be tailored to the host environment. Specific corporate security policies for servers can be implemented, with different filters for servers used for different application.
• Protection is provided independent of topology. Thus both internal and external attacks must pass through the firewall.
• Used in conjunction with stand-alone firewalls, the host-based firewall provides an additional layer of protection. A new type of server can be added to the network, with its own firewall, without the necessity of altering the network firewall configuration.

9.11 Between internal and external firewalls are one or more networked devices in a region referred to as a DMZ (demilitarized zone) network. Systems that are externally accessible but need some protections are usually located on DMZ networks. Typically, the systems in the DMZ require or foster external connectivity, such as a corporate Web site, an e-mail server, or a DNS (domain name system) server.

9.12 An **external firewall** is placed at the edge of a local or enterprise network, just inside the boundary router that connects to the Internet or some wide area network (WAN). One or more **internal firewalls** protect the bulk of the enterprise network.

9.13 An IPS blocks traffic, as a firewall does, but makes use of the types of algorithms developed for IDSs.

9.14 Like an IDS, an IPS can be host-based, network-based, or distributed/hybrid combining information from a range of host and network based sensors.

9.15 Once an IDS has detected malicious activity, it can respond by modifying or blocking network packets across a perimeter or into a host, or by modifying or blocking system calls by programs running on a host.

9.14 A UTM performs network firewalling, network intrusion detection and prevention and gateway anti-virus.

ANSWERS TO PROBLEMS

9.1 It will be impossible for the destination host to complete reassembly of the packet if the first fragment is missing, and therefore the entire packet will be discarded by the destination after a time-out.

9.2 When a TCP packet is fragmented so as to force interesting header fields out of the zero-offset fragment, there must exist a fragment with FO equal to 1. If a packet with FO = 1 is seen, conversely, it could indicate the presence, in the fragment set, of a zero-offset fragment with a transport header length of eight octets Discarding this one-offset fragment will block reassembly at the receiving host and be as effective as the direct method described above.

9.3 If the router's filtering module enforces a minimum fragment offset for fragments that have non-zero offsets, it can prevent overlaps in filter parameter regions of the transport headers.

9.4
1. Allow return TCP Connections to internal subnet.
2. Prevent Firewall system itself from directly connecting to anything.
3. Prevent External users from directly accessing the Firewall system.
4. Internal Users can access External servers,
5. Allow External Users to send email in.
6. Allow External Users to access WWW server.
7. Everything not previously allowed is explicitly denied.

9.5
a. Rules A and B allow inbound SMTP connections (incoming email)
Rules C and D allow outbound SMTP connections (outgoing email)
Rule E is the default rule that applies if the other rules do not apply.
b. Packet 1: Permit (A); Packet 2: Permit (B): Packet 3: Permit (C)
Packet 4: Permit (D)

c. The attack could succeed because in the original filter set, rules B and D allow all connections where both ends are using ports above 1023.

9.6 **a.** A source port is added to the rule set.

b. Packet 1: Permit (A); Packet 2: Permit (B): Packet 3: Permit (C) Packet 4: Permit (D); Packet 5: Deny (E); Packet 6: Deny (E)

9.7 **a.** Packet 7 is admitted under rule D. Packet 8 is admitted under rule C.

b. Add a column called ACK Set, with the following values for each rule:
A = Yes;
B = Yes; C = Any; D = Yes; E = Any

9.8

	Pros	Cons
Pattern matching	•Identifies known attacks •Provides specific information for analysis and response	•May trigger false positives •Requires frequent updates of signature tables •Attacks can be modified to avoid detection
Stateful matching	•Identifies known attacks •Detects signatures spread across multiple packets •Provides specific information for analysis and response	•May trigger false positives • Requires frequent updates of signature tables •Attacks can be modified to avoid detection
Protocol anomaly	•Can identify attacks without a signature •Reduces false positives with well-understood protocols	•May lead to false positives and false negatives with poorly understood or complex protocols •Protocol analysis modules take longer to deploy to customers than signatures
Traffic anomaly	•Can identify unknown attacks and DoS floods	•Can be difficult to tune properly •Must have clear understanding of normal traffic environment
Statistical anomaly	•Can identify unknown attacks and DoS floods	•Can be difficult to tune properly •Must have clear understanding of normal traffic environment

9.9 A requirement like "all external Web traffic must flow via the organization's Web proxy." is easier stated than implemented. This is because identifying what actually constitutes “web traffic” is highly problematical. Although the standard port for HTTP web servers is port 80, servers are found on a large number of other ports (including servers belonging to large, well-known and widely used organizations). This means it is very difficult to block direct access to all possible web servers just using port filters. Whilst it is easy enough to configure web browser programs to always use a proxy, this will not stop direct access by other programs. It also means that the proxy server must have access to a very large number of external ports, since otherwise access to some servers would be limited. As well as HTTP access, other protocols are used on the web. All of these should also be directed via the proxy in order to implement the desired policy. But this may impact the operation of other programs using these protocols. In particular, the

HTTPS protocol is used for secure web access that encrypts all traffic flowing between the client and the server. Since the traffic is encrypted, it means the proxy cannot inspect its contents in order to apply malware, SPAM or other desired filtering. Whilst there are some mechanisms for terminating the encrypted connections at the proxy, they have limitations and require the use of suitable browsers and proxy servers.

9.10 A possible requirement to manage information leakage requires all external e-mail to be given a sensitivity tag (or classification) in its subject and for external e-mail to have the lowest sensitivity tag. At its simplest a policy can just require user's to always include such a tag in email messages. Alternatively with suitable email agent programs it may be possible to enforce the prompting for and inclusion of such a tag on message creation. Then, when external email is being relayed through the firewall, the mail relay server must check that the correct tag value is present in the Subject header, and refuse to forward the email outside the organization if not, and notify the user of its rejection.

9.11 Suitable packet filter rulesets FOR the "External Firewall" and the "Internal Firewall" respectively, to satisfy the stated "informal firewall policy", could be:

action	src	port	dest	port	flags	comment
permit	DMZ mail gateway	any	any	SMTP (25)		header sanitize
permit	any	any	DMZ mail gateway	SMTP (25)		content filtered
permit	any	any	DMZ mail gateway	POP3S (995)		user auth
permit	DMZ web proxy	any	any	HTTP/S (80,443)		content filtered, user auth
permit	DMZ DNS server	DNS (53)	any	DNS (53)		TCP & UDP
permit	any	DNS (53)	DMZ DNS server	DNS (53)		TCP & UDP
permit	any	any	any DMZ server	any	estab lished	return traffic flow
deny	any	any	any	any		block all else

action	src	port	dest	port	flags	comment
permit	any internal	any	DMZ mail gateway	SMTP (25)		
permit	any internal	any	DMZ mail gateway	POP3/S (110,995)		user auth
permit	any internal	any	DMZ web proxy	HTTP/S (80,443)		content filtered, user auth
permit	any internal	DNS (53)	DMZ DNS server	DNS (53)		UDP lookup
permit	DMZ DNS server	DNS (53)	any internal	DNS (53)		UDP lookup
permit	any internal	any	any DMZ server	SSH (22)		user auth on server
permit	mgmt user hosts	any	any DMZ server	SNMP (161)		
permit	any DMZ server	any	mgmt user hosts	SNMP TRAP (162)		
permit	any DMZ server	any	any internal	any	estab lished	return traffic flow
deny	any	any	any	any		block all else

9.12 Yes. A rule set such as the following will do the trick:

```
drop tcp *:* -> 5.6.7.8:*
```

The following might be a little better, because it does not restrict outbound connections initiated by our internal server:

```
drop tcp *:* -> 5.6.7.8:* (if SYN flag set)
```

9.13 a. Here are the strengths.

(1) It mediates all incoming traffic from external hosts and can protect against many attacks by outsiders;
(2) It is easier to manage and to update policies, because of single central location;
(3) It protects against some kinds of DoS attacks launched from the outside.

Here are the weaknesses.
(1) It has no protection against malicious insiders;
(2) It has no protection for mobile laptops while they are connected to other networks;
(3) It has no protection if laptops get infected while travelling and then spread infection when they re-connect to our internal network

b. Here are the strengths.
(1) It protects against malicious insiders and infected internal machines as well as outside attackers;
(2) It protects laptops even while they are travelling and connected to other networks;
(3) It may be easier to customize firewall protection on a per-machine basis.

Here are the weaknesses.
(1) It is potentially more difficult to manage policies, due to the number of machines whose rulesets must be configured and updated;
(2) Uncooperative users may be able to modify settings or disable firewalls on their own machines, and viruses/worms may be able to do the same to machines they infect;
(3) It is potentially less resistant to DDoS, since DoS attacks can still flood internal network links;
(4) Depending upon firewall configuration, it may block legitimate internal traffic and/or make some internal services harder to use.

c. Here are the strengths.
(1) Layered defense provides redundancy in case one firewall fails;
(2) It can easily update policy against external attacks if a new threat develops, which gives some time to update the rulesets on internal hosts.
(3) Strengths (a)(1) and (b)(1)–(3) also apply.

Here are the weaknesses.
(1) Potential for overblocking of legitimate traffic, since traffic flows only if permitted by both firewalls.
(2) Weaknesses (b)(1), (b)(4) also apply

9.14 Modify the rule action from `Alert` to `Drop` to block these packets entering the home network, and to log the attempt:

```
Drop tcp $EXTERNAL_NET any -> $HOME_NET any\
(msg: "SCAN SYN FIN" flags: SF, 12;\
reference: arachnids, 198; classtype: attempted-recon;)
```

CHAPTER 10 BUFFER OVERFLOW

ANSWERS TO QUESTIONS

10.1 A "buffer overflow" results from adding more information to a program's buffer than it was designed to hold.

10.2 Buffer overflow attacks typically target buffers located in one of the stack, the heap, or the data section of a process.

10.3 The consequences of a buffer overflow include corruption of data used by the program, unexpected transfer of control in the program, possibly memory access violations, and very likely eventual program termination. When done deliberately as part of an attack on a system, the transfer of control could be to code of the attacker's choosing, resulting in the ability to execute arbitrary code with the privileges of the attacked process.

10.4 To exploit any type of buffer overflow, the attacker needs to identify both a buffer overflow vulnerability in some program that can be triggered using externally sourced data under the attackers control, and to understand how that buffer will be stored in the processes memory, and hence the potential for corrupting adjacent memory locations and potentially altering the flow of execution of the program.

10.5 The programming languages vulnerable to buffer overflows are those without a very strong notion of the type of variables, and what constitutes permissible operations on them. They include assembly language, and C and similar languages. Strongly typed languages such as Java, ADA, Python, and many others are not vulnerable to these attacks.

10.6 A "stack buffer overflow" occurs when the targeted buffer is located on the stack, usually as a local variable in a function's stack frame. If the function code that copies externally sourced data into such a buffer fails to correctly limit the amount of data written, then the values of adjacent variables, and even control fields such as the saved frame pointer and return address, can be overwritten. This can result in the program crashing, or in execution being transferred to (shell) code the attacker provides.

10.7 **"**shellcode" is code supplied by the attacker, and often saved in the buffer being overflowed. It is called **"**shellcode" because traditionally its function was to transfer control to a user command-line interpreter, or shell, which gave access to any program available on the system with the privileges of the attacked program.

10.8 There are several generic restrictions on the content of shellcode. Firstly it has to be "position-independent". That means it cannot contain any absolute address referring to itself, because the attacker generally cannot determine in advance exactly where the targeted buffer will be located in the stack frame of the function in which it is defined. Instead the code is written to determine its location when actually run. Another restriction is that it cannot contain any NULL values. This is a consequence of the common use of C string routines to copy this data into the buffer. To overcome this, any NULL values must be written in when the code actually runs.

10.9 A "NOP sled" is a run of NOP (no operation, do nothing) instructions, which are included before the desired shellcode to help overcome the lack of knowledge by the attacker of its precise location. In a buffer overflow attack, the attacker arranges for the transfer of control (via overwritten return address) to occur somewhere in the NOP Sled (guessing around the middle of the most likely location). Then when control transfers, no matter where in this run it occurs, the CPU executes NOPs until it reaches the actual desired shellcode.

10.10 Apart from just spawning a command-line shell, the attacker may wish to create shellcode to perform somewhat more complex operations. The Packet Storm website includes a large collection of packaged shellcode, including code that can: set up a listening service to launch a remote shell when connected to; create a reverse shell that connects back to the hacker; local exploits that establish a shell or execve a process; flush firewall rules (such as IPTables and IPChains) that currently block other attacks; break out of a chrooted (restricted execution) environment, giving full access to the system.

10.11 Two broad categories of defenses against buffer overflows are: compile-time defenses which aim to harden programs to resist attacks in new programs; and run-time defenses which aim to detect and abort attacks in existing programs.

10.12 Compile-time defenses include: writing programs using a modern high-level programming language that is not vulnerable to buffer overflow attacks; using safe coding techniques to validate buffer use;

using language safety extensions and/or safe library implementations; or using stack protection mechanisms.

10.13 Run-time defenses that provide some protection for existing vulnerable programs include: using "Executable Address Space Protection" that blocks execution of code on the stack, heap, or in global data; using "Address Space Randomization" to manipulate the location of key data structures such as the stack and heap in the processes address space; or by placing **guard pages** between critical regions of memory in a processes address space.

10.14 In a "return to system call" attack, typically a stack overflow is used, but the return address is changed to jump to existing code on the system, typically in standard libraries. This avoids triggering run-time defenses that block executable code on the stack or heap. They can be prevented using stack protection mechanisms.

10.15 In a "heap buffer overflow" attack, the targeted buffer is located on the heap rather than stack. These generally target either function pointers located adjacent to such buffers, or heap management pointers. These attacks avoid defenses that focus on stack based attacks. Defenses include making the heap non-executable, or randomizing addresses of structures on the heap.

10.16 In a "global data area overflow" attack, the targeted buffer is located in the global data area. The attack is similar to heap overflows, as are the defenses.

ANSWERS TO PROBLEMS

10.1 Safer variants of the unsafe standard C library functions shown in Table 10.2 are:

Original Unsafe Function	Safer Alternative
`gets(char *str)`	`fgets(char *str, int size, FILE * fil)`
`sprintf(char *str, char *fmt, ...)`	`snprintf(char *str, int size, char * fmt, ...)`
`strcat(char *dest, char *src)`	`strncat(char *dest, char *src, int count)` `strlcat(char *dest, char *src, int size)`
`strcpy(char *dest, char *src)`	`strncpy(char *dest, char *src, int count)` `strlcpy(char *dest, char *src, int size)`
`vsprintf(char *str, char *fmt, va_list ap)`	`vsnprintf(char *str, int size, char *fmt, va_list ap)`

nb. the `strlXXX` routines are regarded as safer than the `strnXXX` routines, but may not be available on all systems.

10.2 Corrected version of the program shown in Figure 10.1a (see bold lines):

```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    fgets(str2, sizeof(str2), stdin);
    if (strncmp(str1, str2, sizeof(str2)) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

10.3 Corrected version of the program shown in Figure 10.5a (see bold lines):

```
void hello(char *tag)
{
    char inp[16];

    printf("Enter value for %s: ", tag);
    fgets(inp, sizeof(inp), stdin);
    printf("Hello your %s is %s\n", tag, inp);
}
```

10.4 Corrected version of (part of) the program shown in Figure 10.7a (see bold lines):

```
void display(char *val)
{
    char tmp[16];
    snprintf(tmp, sizeof(tmp), "read val: %s\n", val);
    puts(tmp);
}
```

10.5 The extended shellcode from Figure 10.8b including a call to `exit(0)` is (see bold lines):

```
        jmp    find              // jump to end of code
cont:   pop    %esi              // pop address of sh off stack into %esi
        xor    %eax,%eax         // zero contents of EAX
        mov    %al,0x7(%esi)     // copy zero byte to end of string sh (%esi)
        lea    (%esi),%ebx       // load address of sh (%esi) into %ebx
        mov    %ebx,0x8(%esi)    // save address of sh in args[0] (%esi+8)
        mov    %eax,0xc(%esi)    // copy zero to args[1] (%esi+c)
        mov    $0xb,%al          // copy execve syscall number (11) to AL
        mov    %esi,%ebx         // copy address of sh (%esi) t0 %ebx
        lea    0x8(%esi),%ecx    // copy address of args (%esi+8) to %ecx
        lea    0xc(%esi),%edx    // copy address of args[1] (%esi+c) to %edx
        int    $0x80             // software interrupt to execute syscall
        mov    $0x1,%al          // copy exit syscall number (1) to AL
        xor    %ebx,%ebx         // zero contents of EBX
        int    $0x80             // software interrupt to execute syscall
find:   call   cont              // call cont which saves next address on
stack
sh:     .string "/bin/sh "       // string constant
args:   .long 0                  // space used for args array
        .long 0                  // args[1] and also NULL for env array
```

Note that the syscall numbers are listed in the architecture specific "unistd.h" include file. This code can be tested by changing the encoded shell name to one that does not exist (e.g.. "/bin/xx").

10.6 The details and results from running this experiment depend on the specific UNIX O/S variant and example vulnerable program used. The book's Web site includes a zipfile with the example programs used in this chapter, as run on a Knoppix CD-bootable system. Note that you may well need to use an older O/S release, since recent versions have defenses such as non-executable stack enabled by default.

10.7 The **Packet Storm (**http://www.packetstormsecurity.org/**)** site (shellcode area) includes several examples of PPC code to exec a shell under MacOSX (http://www.packetstormsecurity.org/shellcode/execMacOSX.txt, http://www.packetstormsecurity.org/shellcode/osx72bytes.txt) and Linux/PPC

(http://www.packetstormsecurity.org/shellcode/execve-core.c). It also includes a comprehensive paper on this topic http://www.packetstormsecurity.org/shellcode/PPC_OSX_Shellcode_Assembly.pdf.

10.8 The details depend on which safe library alternative is examined. Note the Wikipedia page on "Buffer_overflow" provides links to the websites for many of the alternative safe library implementations.

10.9 No detailed answer available yet, however the Wikipedia page on "Return-to-libc_attack" provides some brief information and links to other resources.

10.10 Corrected version of the functions shown in Figure 10.10 (see bold lines). Note that the function "signatures" have to change, since information on the size of the buffer is needed (as seen in the safer variants of the string copy/cat functions).

```
int safe_copy_buf(char *to, int size, int pos, char *from, int len)
{
    int i;

    if (len <= 0)         /* invalid negative or zero len */
        return pos;
    if ((pos+len)>size)) /* len exceeds available space in buffer */
        len = size – pos;
    for (i=0; i<len; i++) {
        to[pos] = from[i];
        pos++;
    }
    return pos;
}
```

```
short safe_read_chunk(FILE fil, int size, char *to)
{
    short len;
    fread(&len, 2, 1, fil);     /* read length of binary data */
    if (len <= 0)               /* invalid negative or zero len */
        return 0;
    if (len > size))            /* len exceeds space in buffer */
        len = size;
    fread(to, 1, len, fil);     /* read len bytes of binary data
    return len;
}
```

10.11 Corrected version of the program shown in Figure 10.11a (see bold lines):

```
#define INP_SIZE 64
/* record type to allocate on heap */
typedef struct chunk {
    char inp[INP_SIZE];         /* input buffer */
    void (*process)(char *);    /* pointer to function to process inp */
} chunk_t;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer5 read %d chars\n", len);
}

int main(int argc, char *argv[])
{
    chunk_t *next;

    setbuf(stdin, NULL);
    next = malloc(sizeof(chunk_t));
    next->process = showlen;
    printf("Enter value: ");
    fgets(next->inp, INP_SIZE, stdin);
    next->process(next->inp);
    printf("buffer5 done\n");
}
```

10.12 The details depend on the current vulnerability status.

10.13 No detailed answer available yet, however [LHEE03] includes details on the “format string overflow” attack.

10.14 No detailed answer available yet, however the Wikipedia page on “Integer overflow” includes some information on the “integer string overflow” attack.

CHAPTER 11 SOFTWARE SECURITY

ANSWERS TO QUESTIONS

11.1 Software quality and reliability is concerned with the accidental failure of a program as a result of some theoretically random, unanticipated input, system interaction, or use of incorrect code. These failures are expected to follow some form of probability distribution. Software security differs in that the attacker chooses the probability distribution, targeting specific bugs that result in a failure that can be exploited by the attacker. These bugs may often be triggered by inputs that differ dramatically from what is usually expected, and hence are unlikely to be identified by common testing approaches.

11.2 **Defensive programming** is a form of defensive design intended to ensure the continuing function of a piece of software in spite of unforeseeable usage of said software. The idea can be viewed as reducing or eliminating the prospect of Murphy's Law having effect. Defensive programming techniques come into their own when a piece of software could be misused mischievously or inadvertently to catastrophic effect.

11.3 Program input refers to **any** source of data that originates outside the program, and whose value is not explicitly known by the programmer when the code was written. It includes data read into the program from user keyboard or mouse entry, files, or network connections. It also includes data supplied to the program in the execution environment, the values of any configuration or other data read from files by the program, and values supplied by the operating system to the program.

11.4 An **injection attack** refers to a wide variety of program flaws related to invalid handling of input data, particularly when such input data can accidentally or deliberately influence the flow of execution of the program. Examples of injection attacks include: command injection, SQL injection, code injection, and remote code injection. There are a wide variety of mechanisms that can result in injection attacks. These include when input data is passed as a parameter to another helper program (command) or to a database system (SQL), whose output is then processed and used by the original program. Or when the input

includes either machine or script code that is then executed/interpreted by the attacked system (code).

11.5 In a **command injection** attack, the unchecked input is used in the construction of a command that is subsequently executed by the system with the privileges of the attacked program. In an SQL injection attack, the user-supplied input is used to construct a SQL request to retrieve information from a database. In both cases the unchecked input allows the execution of arbitrary programs/SQL queries rather than the program/query specified by the program designer. They differ in the syntax of the respective shell/SQL meta-characters used that allow this to occur.

11.6 A cross-site scripting attack occurs when concerns input provided to a program by one user, is subsequently output to another user. They are most commonly seen in scripted web applications, where the vulnerability involves the inclusion of script code in the HTML content of a web page displayed by a user's browser. An example of such an attack concerns an unsafe guestbook application where a comment saved by one user includes javascript code, such as that shown in Figure 11.5a, that executes when the comment is viewed by other users.

11.7 The main technique used by a defensive programmer to validate assumptions about program input is to compare it against a regular expressions, which is a pattern that describes either what is wanted or what is known to be dangerous. The result of the comparison is used to either accept wanted, or reject dangerous, input.

11.8 When using the Unicode character set a further issue concerns multiple, alternative, redundant encodings of the input data which Unicode allows. Given the possibility of multiple encodings, the input data must first be transformed (canonicalized) into a single, standard, minimal representation. This involves replacing alternate, equivalent encodings by one common value. Then the input data can be compared to a single representation of acceptable input values.

11.9 "Input fuzzing" is a software testing technique that uses very large amounts of randomly generated data as inputs to a program, to determine whether the program or function correctly handles all such abnormal inputs, or whether it crashes or otherwise fails to respond appropriately. The major advantage of fuzzing is its simplicity, low cost, and its freedom from assumptions about the "expected" input to any program, service or function. It ought to be deployed as a component of any reasonably comprehensive testing strategy, especially in relation to commonly deployed software.

11.10 Key software security concerns associated writing safe program code include whether the implemented algorithm correctly solves the specified problem, whether the machine instructions executed correctly represent the high-level algorithm specification, and whether the manipulation of data values in variables, as stored in machine registers or memory, is valid and meaningful.

11.11 A race condition can occur when several processes, or threads within a process, simultaneously access the same shared memory without suitable synchronization. The result can be that the shared memory values may be corrupted, or changes lost, due to overlapping access, use and replacement of the shared values.

11.12 Environment variables are a collection of string values inherited by each process from its parent, that can affect the way a running process behaves. The operating system includes these in the processes memory when it is constructed. Well known environment variables include the variable `PATH` which specifies the set of directories to search for any given command, `IFS` which specifies the word boundaries in a shell script, and `LD_LIBRARY_PATH` which specifies the list of directories to search for dynamically loadable libraries. All of these have been used to attack programs, and especially privileged shell scripts. The attacker changes the values of one or more of these, then calls a script running with other (higher) privileges, which is then "tricked" into running a program or loading a library of the attackers choice as a result.

11.13 The principle of least privilege states that programs should execute with the least amount of privileges needed to complete their function.

11.14 There are several issues associated with the correct creation and use of a lockfile. Firstly it is purely advisory, since all programs using this form of synchronization must cooperate. A more serious flaw can occur in its implementation, if it fails to atomically both check that the lockfile does not exist, and also then create it.

11.15 There are several issues associated with the correct creation and use of a temporary file in a shared directory, as they must be both unique, and not accessed by other processes. An attacker may attempt to guess the temporary filename a privileged program will use, and then attempt to create their own version in the interval between the program checking the file does not exist, and subsequently creating it. Secure temporary file creation and use requires the use of a random temporary filename, and its checking

and creation using an atomic system primitive, similar to the creation of a lockfile.

11.16 Problems that may result from a program sending unvalidated input from one user to another user if the output does not conform to the expected form and interpretation by the recipient. A program may accept input from one user, save it, and subsequently display it to another user. If this input contains content that alters the behavior of the program or device displaying this data, and it is not adequately sanitized by the program, then an attack on the user is possible. Examples include embedding terminal (e.g.. VT100) "escape sequences", or Javascript script code in an XSS attack.

ANSWERS TO PROBLEMS

11.1 Information on writing regular expressions or patterns may be found in just about any good text on using UNIX systems. The Wikipedia page on "Regular_expression" provides a reasonable amount of detail, and pointers to other references. Texts on languages with regular expression support, such as perl, php or python, should be consulted for the variants they accept.

11.2 Meta-characters used by the Linux/UNIX Bourne shell include:

```
; & | ( ) { } < > * ? [ ] ~ ! " ' \ `
```

In particular the '; && || |' characters are variously used to separate commands, and would most likely be seen in any attack. Other common shells such as BASH or CSH have the same interpretation of these characters, although they treat others differently. Hence input validation checks to prevent command injection attacks for all of these shells must typically reject the same characters (although as noted in the text, it is much better to write patterns to accept known good input values, which typically are alphanumeric plus limited punctuation).

11.3 Rewritten, extended version of perl script shown in Figure 11.2 (see bold lines):

```
#!/usr/bin/perl
# finger.cgi - finger CGI script using Perl5 CGI module

use CGI;
use CGI::Carp qw(fatalsToBrowser);
$q = new CGI;

# display HTML header
print $q->header,
      $q->start_html('Finger User'),
      $q->h1('Finger User');

# get and validate the name of user
$user = $q->param("user");
showError("The specified user '$user' contains illegal characters!")
    unless (("$user" =~ /^\w[-+%\w\t ]*\w$/) || ("$user" =~ /^\w$/));

# obtain desired finger information (including error messages)
print "<pre>";
print `/usr/bin/finger -sh "$user" 2>&1`;

# display HTML footer
print "</pre>";
print $q->end_html;
exit(0);

# ------------------------------------------------------------
# subroutine showError(reason) - build HTML error response
sub showError
{
    local ($msg) = $_[0];       # description of error
    print "<h1>Error</h1>\n";
    print "$msg";
    print "<p>Unable to safely obtain finger information.\n";
    print "<p>Please go back and correct the input supplied.\n";
    print $q->end_html;
    exit(0);
}
```

11.4 There are a number of deficiencies in the script shown in Figure 11.10a. Despite an attempt to quote the values of the supplied form fields subject, from, body, if any of these include a " character, the shell will assume it's the end of the quoted string, and interpret any shell meta-characters in the remainder. This can allow the execution of arbitrary commands, with out being displayed in the response web form. For example, including a value for "Your Email Address" like:

```
user@some.domain"; whoami; ls –al; echo "
```

will result in the response including the output from the commands: whoami; ls –al

A better version of this script, shown in Figure 11.2, with more stringent input checking (though the email address checking is simplified, and the body cannot contain " characters) is:

```
#!/usr/bin/perl
# comment.cgi - send comment to webmaster
# specify recipient of comment email
$to = "webmaster";

use CGI;
use CGI::Carp qw(fatalsToBrowser);
$q = new CGI;       # create query object

# display HTML header
print $q->header,
$q->start_html('Comment Sent'),
$q->h1('Comment Sent');

# retrieve form field values and send comment to webmaster
$subject = $q->param("subject");
$from = $q->param("from");
$body = $q->param("body");

# validate the input information
# subject MUST NOT contain " or multiple lines
showError("The subject '$subject' contains illegal characters!")
    if ("$subject" =~ m:["\r\n]:);
# from MUST only contain characters valid in an email address
showError("The from address '$from' contains illegal characters!")
    unless ("$from" =~ m:^[-_.=/\w][- _.=%!/@\w]*[-_.=%\w]$:);
# body MUST NOT contain "
showError("The body '$body' contains illegal characters!")
    if ("$body" =~ m:":);

# generate and send comment email
system("export REPLYTO=\"$from\"; echo \"$body\" | mail -s \"$subject\"
$to");

# indicate to user that email was sent
print "Thankyou for your comment on $subject.";
print "This has been sent to $to.";

# display HTML footer
print $q->end_html;
exit(0);

# ------------------------------------------------------------
# subroutine showError(reason) - build HTML error response due to reason
sub showError
{
    local ($msg) = $_[0];       # description of error
    print "<h1>Error</h1>\n";
    print "$msg";
    print "<p>Unable to safely send comment.\n";
    print "<p>Please go back and correct the input supplied.\n";
    print $q->end_html;
    exit(0);
}
```

To remove the limitation on the contents of the mail body, the script would need to be further rewritten to open a pipeline to the mail program, and explicitly write the body contents into this pipeline, so they are not interpreted by the shell.

11.5 No short answer is available, as it depends on the scripting language chosen.

11.6 No short answer is available, as it depends on the scripting language chosen.

11.7 No short answer is available, as it requires research to determine the current state of this field. Information on some fuzzing tools is available from the Fuzz Testing of Application Reliability (http://www.cs.wisc.edu/~bart/fuzz/) site.

11.8 No short answer is available, as it requires research to determine the current state of this field.

11.9 No detailed answer is available, as it depends on the system and shell used. The value of all environment variables can be displayed using the "env" command. A variable can be changed temporarily by changing the value of the corresponding shell variable, and then exporting it (details vary depending on the shell used). To change a value permanently for all subsequent logins on the system, the relevant shell startup file, either system-wide, or for a specific user, must be changed. Again the name and location of these files varies depending on the shell and system used.

11.10 No short answer is available, as this question requires experimentation with the supplied scripts.

CHAPTER 12 OPERATING SYSTEM SECURITY

ANSWERS TO QUESTIONS

12.1 The basic steps needed in the process of securing a system (from [SCAR08]) are:
•assess risks and plan the system deployment
•secure the underlying operating system and then the key applications
•ensure any critical content is secured
•ensure appropriate network protection mechanisms are used
•ensure appropriate processes are used to maintain security

12.2 The aim of the specific system installation planning process is to maximize security whilst minimizing costs, which is best done before implementation. It needs to determine the security requirements for the system, its applications and data, and of its users.

12.3 The basic steps needed to secure the base operating system (from [SCAR08]) are:
•install and patch the operating system
•harden and configure the operating system to adequately address the identified security needs of the system by:
    removing unnecessary services, applications, and protocols
    configuring users, groups and permissions
    configuring resource controls
•install and configure additional security controls, such as anti-virus,
•host-based firewalls and IDS, if needed
•test the security of the basic operating system to ensure that the
•steps taken adequately address its security needs

12.4 Keeping all software as up to date as possible so important due to the continuing discovery of software and other vulnerabilities for commonly used operating systems and applications.

12.5 Automated patching:
Pros: minimizes window of opportunity for attackers when new vulnerabilities are found; is convenient, especially if automated.

Cons: patches sometimes introduce instability, especially on change controlled systems.

12.6 The point of removing unnecessary services, applications and protocols is to minimize the amount of software that can run, since if less software is available to run, then the risk that it may contain vulnerabilities is reduced.

12.7 Additional security controls that may be used to secure the base operating system include: anti-virus software, host-based firewalls, IDS or IPS software, and to white-list applications.

12.8 Additional steps are used to secure key applications are:
•to install and patch each application to the most recent supported secure version
•to perform application specific configuration
•enable encryption and generate keys and certificates if required

12.9 The steps are used to maintain system security (from [SCAR08]) are:
•monitoring and analyzing logging information
•performing regular backups
•recovering from security compromises
•regularly testing system security
•using appropriate software maintenance processes to patch and
•update all critical software, and to monitor and revise configuration as needed

12.10 Configuration of applications and services on Unix and Linux systems is most commonly implemented using separate text files for each application and service. System wide configuration details are generally located in either the `/etc` directory, or in the installation tree for a specific application. Where appropriate, individual user configuration that can override the system defaults, are located in hidden "dot" files in each user's home directory.

12.11 Unix and Linux systems implement discretionary access control (DAC) to all file system resources, including not only files and directories, but devices, processes, memory and indeed most system resources.

12.12 Access is specified as granting read, write, and execute permissions to each of owner, group and others, for each resource, as shown in figure 4.6.

12.13 These extended access rights on Unix and Linux systems are typically set and displayed using the **getfacl** and **setfacl** commands.

12.14 Programs that set user (setuid) to some user (even root, the superuser), or set group (setgid) to some group on Unix and Linux systems execute with the specified user's rights, or with access to resources belonging to the group, no matter which user executes them.

12.15 Linux systems primarily now use the **iptables** program to configure the **netfilter** kernel module. This provides comprehensive, though complex, stateful packet filtering, monitoring and modification capabilities.

12.16 Logging can generate significant volumes of information. A suitable automatic log rotation and archive system can be configured to assist in managing the overall size of the logging information.

12.17 Unix and Linux systems provide a mechanism to run services in a **chroot jail**, which restricts the servers view of the file system to just a specified portion, and helps contain the effects of a given service being compromised or hijacked.

12.18 Users and groups in Windows systems may be stored and used locally, on a single system, in the Security Account Manager (SAM). It may also be centrally managed for a group of systems belonging to a domain, with the information supplied by a central Active Directory (AD) system using the LDAP protocol. Most organizations with multiple systems will manage them using domains.

12.19 There are major differences between the implementations of the discretionary access control models on Unix and Linux systems verses Windows systems. Unix and Linux systems use a small number of access rights for subjects being owner/group/other across nearly all resources/objects – this means it is a simple model, but because the same rights are used everywhere, their meaning can differ for different objects, and sometimes it is hard or impossible to specify some desired complex requirements. Windows systems use a much larger set of access rights, which differ for different types of objects – a more complex model, which can be harder to master, but which may allow better specification of some desired complex requirements.

12.20 The mandatory integrity controls used for in Windows systems label all objects, such as processes and files, and all users, as being of low, medium, high or system integrity level. Then whenever data is written to an object, the system first ensures that the subject's

integrity is equal or higher than the object's level, implementing a form of the Biba Integrity model.

12.21 On Windows, the privilege that overrides all ACL checks is the ability to backup the computer, which requires over-riding the normal access controls to obtain a complete backup.

12.22 On Windows systems, much of the application and service configuration information is centralized in the Registry, which forms a database of keys and values that may be queried and interpreted by applications on these systems.

12.23 Virtualization refers to a technology that provides an abstraction of the computing resources used by some software, which thus runs in a simulated environment called a virtual machine (VM).

12.24 Full virtualization, which allows multiple full operating system instances to execute on virtual hardware, supported by a hypervisor that manages access to the actual physical hardware resources. Specifically we discuss securing both native and hosted virtualization variants of this.

12.25 The main security concerns with virtualized systems (from [SCAR11]) are:

•guest OS isolation, ensuring that programs executing within a guest OS may only access and use the resources allocated to it, and not covertly interact with programs or data in either other guest OS's or in the hypervisor.

•guest OS monitoring by the hypervisor, which has privileged access to the programs and data in each guest OS, and must be trusted as secure from subversion and compromised use of this access

•virtualized environment security, particularly as regards image and snapshot management, which attackers may attempt to view or modify

12.26 The basic steps to secure virtualized systems (from [NIST11]) are:

•carefully plan the security of the virtualized system

•secure all elements of a full virtualization solution, including the hypervisor, guest O/S's, and virtualized infrastructure; and maintain their security

•ensure that the hypervisor is properly secured

•restrict and protect administrator access to the virtualization solution

ANSWERS TO PROBLEMS

12.1 If a process running as root is compromised, then any child-processes the attacker spawns (such as a remote shell) will also run as root. If such a process can be used to read data, it will be able to read any local file. If such a process spawns a process that listens on UDP or TCP ports, that process can be bound to privileged ports (TCP 22, TCP 80, and all other TCP/UDP ports lower than 1024). If such a process is merely sloppily coded, the impact of bug behavior may be much greater than if the process didn't run as root (overwriting important files, interfering with other processes, etc.).

12.2 The 'find' command allows for searches on the permissions, using the 'perm' option, '-perm +4000' will find all files with the SUID bit set. The full command, with output to file, is:

```
find / -perm +4000 > somelogfile
```

To find both SUID and SGID files (and to show the symbolic form of perms) you can use:

```
find / -perm +u+s -o -perm +g+s > somelogfile
```

Once a list of files has been generated, the system operator can check the modification time and other statistics (like size, owner) to ensure the file is the genuine article. However, this is still a manual process, prone to human error. System Security Tools (like "tripwire") could be applied to the files in the list to actively monitor for modifications.

12.3 Filesystem permissions are important because practically everything in Linux is represented by some sort of file. They specify whether a given user or group-member (**subject**) can read, write, or execute (**actions**) a given file, directory, special file, symbolic link, or other filesystem element (**objects**).

12.4

```
drwxr-x---    2 ahmed staff    0 Jul 21 07:58 stuff
-rw-rw----    1 ahmed staff    0 Jul 21 08:00 ourstuff
```

NOTE: it's acceptable (albeit redundant) for the sticky bit to be set on "stuff," so long as the write bit is not. If you don't want group-users to *either* create or delete files in a directory, it's sufficient to simply turn off the write bit.

12.5 Assets: website availability, system availability, local network integrity (integrity of other systems the attacker may reach via compromised web server), corporate data, customer data, website availability, e-commerce business activity (immediate revenue), company reputation (future revenue)
Vulnerability: buffer-overflow in Apache

Attack-vector: the worm "WorminatorX" (that multiple exploits may target the same vulnerability -- this is just the one we *know* about)
Attackers: competitors, thieves, identity thieves, website defacers (vandals), disgruntled ex-employees, the Byelorusan mob, etc. -- public web sites can be prey to any Internet-connected type of attacker
Likelihood of Occurrence: High (or synonyms thereof) -- Internet worms spread very far very quickly
Likely Impact: High -- complete exposure/loss of any or all affected assets to any potential attacker
Plausible Mitigations: Patch the Apache vulnerability; if no patch is available, protect Apache with SELinux or AppArmor; or run Apache in a chroot jail (Note that Apache already runs as an unprivileged user, by default -- presumably the WorminatorX vulnerability depends on some other privilege-escalation vulnerability)

12.6 Logs provide audit trails of system and application events, and are useful for identifying problems, analyzing security breaches, analyzing system/application failures. (Bonus points if student mentions that in regulated or otherwise controlled environments, logs are usually mandated by industry or governmental auditors.) Logs may even, if monitored closely, provide an early warning of failures or attacks in progress. But logs only capture the level of detail

12.7 'Normal' behavior would generally involve users creating, using or deleting files belonging to either the individual user or to a group to which they belong. Normal behavior would not involve attempting to gain superuser or root privileges or in any other way altering the operating system or attempting to perform what could be considered administrator functions. In particular the rules should watch for:

- bad or repeated login attempts
- copying large numbers of files to either external media or remote locations
- attempts to access system files or log files;
- accessing directories, files or programs that are not usually accessed;
- changing security settings.
- attempts to become a superuser using the su or sudo commands.

A couple of SWATCH examples are:

watchfor /failed/ # echo bold # mail addressess=root,subject=Failed Authentication

watchfor /su:/ # echo bold # mail addresses=root,subject=Someone sued to root access

12.8 The key advantage of file integrity checking tools is that they can precisely locate potentially damaging changes in a filesystem arising

from deviations in expected system behavior, without the need to explicitly codify the kinds of events that might produce such changes.

In the context of intrusion detection, this means that an attempted attack can be flagged by its effect on the integrity of particular system files and not by any a priori knowledge of the behavioral characteristics of the attack. Such information is required to create an meaningful attack signature in signature-based intrusion detection systems, and since new attack patterns cannot be detected until they have been codified, the corpus of known signatures needs to be continually updated. Further, integrity based tools can also detect aberrant system behavior that has nothing to do with malicious activity, such as misbehaving programs and inadvertent user actions that damage files and directories.

The problem with the file-integrity tools (and indeed all anomaly-based systems) is that it is often very difficult to characterize normal system conditions with sufficient accuracy to allow intrusive or aberrant activity to be clearly distinguished. For usability, the rate of false alarms needs to be low, or system administrators will be continually responding to normal or unusual activity that is otherwise perfectly legitimate. The other problem is that integrity checking only provides evidence of malicious or aberrant activity after the fact. It does nothing to prevent it, and you still need effective recovery strategies as an adjunct to the intrusion detection process.

In general, it is desirable that executables (especially those in /bin and /usr/sbin) and critical system information resources (e.g. password files, configuration files for core services, and so on) be stringently monitored. These should not change except as a result of authorized system administrative action. Similarly, logfiles should be monitored to ensure that they are not modified or truncated. Directories, such as /root and /home should be carefully monitored for unauthorized changes, such as additions or modification to permissions. However, it is simply not sensible or feasible to monitor all files in this way, particularly those that change frequently as shared documents and user home accounts.

12.9 Unix and Linux systems use a small number of access rights for subjects being owner/group/other across nearly all resources/objects – this means it is a simple model, but because the same rights are used everywhere, their meaning can differ for different objects, and sometimes it is hard or impossible to specify some desired complex requirements.

Windows systems use a much larger set of access rights, which differ for different types of objects – a more complex model, which can be harder to master, but which may allow better specification of some desired complex requirements.

An increase in security features is desirable provided the system administrator is competent and completely understands the purpose and use of each of the security features, how they interact with each other, and which feature is best used in specific circumstances.

If the administrator was not as familiar with all the complex security features then there would be a benefit in having less security features available. This is because having less to understand means less chance of making mistakes.

12.10 When using BitLocker on a laptop, the laptop should not use standby mode, rather it should use hibernate mode. This is because Hibernate writes memory to the computer's disk drive, which means the computer's memory content, is protected by bitlocker. Standby (aka sleep) simply keeps the computer in a very low power state, and memory is maintained and not protected by BitLocker.