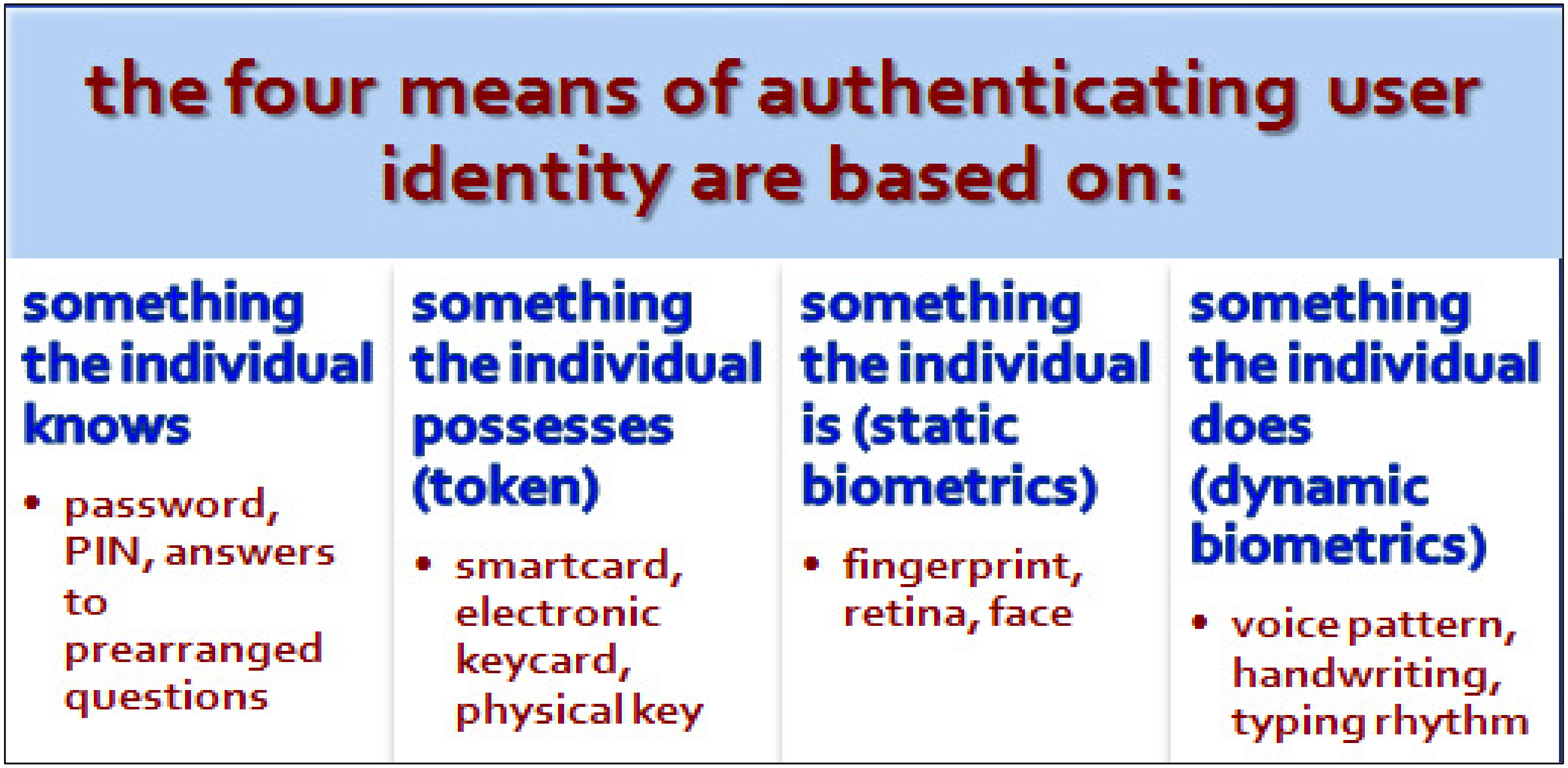
Module 2: Systems Security

Dr. Natarajan Meghanathan
Associate Professor of Computer Science
Jackson State University, MS
E-mail: natarajan.meghanathan@jsums.edu

Topics

2.1 Authentication

- Identification – the process of asserting who a user is.
- Authentication – is the process of determining whether a user should be allowed to access a system (a yes or no decision).



the four means of authenticating user identity are based on:

something the individual knows	something the individual possesses (token)	something the individual is (static biometrics)	something the individual does (dynamic biometrics)
• password, PIN, answers to prearranged questions	• smartcard, electronic keycard, physical key	• fingerprint, retina, face	• voice pattern, handwriting, typing rhythm

Password-based Authentication

- User submits an identifier (ID) and password.
  - The system compares the user entered password to a previously stored password (or a hash) for that user ID, maintained in a system password file.

- The non-randomness in password selection is the root cause of problems with passwords (being cracked easily).
- For a password of 8-characters long chosen from a 256-character set, it may appear that the search space is $256^8 = 2^{64}$.
  - However, a user is far more likely to select an 8-character dictionary word as password (i.e., something that is easy to remember) rather than choosing an arbitrary combination of 8 characters.

- Common Vulnerabilities of Password-based authentication
  - **Offline dictionary attack**: Comparing the hash values of passwords in the system file with those corresponding to commonly used passwords; and extracting the user ID/password
    - A pre-computed table of hashed passwords of possible passwords is called a **rainbow table**.
  - Guess password for a specific account through repeated login attempts (thru' info about user and system policies)
  - **Social Engineering attacks** – tricking users/admin to reveal password

Passwords and Passphrases

- Thumb rule for selecting passwords: Easy to Remember, but difficult to guess. For example,
  - FCa7Yago: Four Centuries and Seven Years ago
  - 1400JLSt: 1400 John Lynch Street
- One should also choose strong password recovery mechanisms with security questions that can be answered only by the user and not easily by a hacker.
- Passphrases are nowadays being considered as an alternative for passwords:
  - A passphrase is a sequence of words or text typically longer than a password
  - Example: `Five Multiplied by Eight: 4Zero`
  - A passphrase is difficult to crack than a password. Even if a passphrase is formed from words that are part of a dictionary, the presence of more than one word in the passphrase leads to an exhaustive search space.

Password Strength

- Password strength – is a measure of the effectiveness of a password to resist guessing and brute-force attacks
  - Quantified in terms of the number of trials an attacker would need to crack the password
  - Password strength is a function of length, complexity and unpredictability.

- Password strength is quantified as the “Number of Entropy bits” (H) and 2^H is the number of attempts an attacker would need to exhaust all possibilities during a brute-force search.

- If a password can be ‘L’ symbols long and the number of possible symbols is ‘N’, the maximum number of passwords is N^L, if any combination of *L* symbols could be a password. In that case, the password entropy H (# bits) = $\log_2 N^L = L*\log_2 N$.
- ➔Entropy (# bits) per symbol of the password = $\log_2 N$.

Password Strength

- Password strength, in terms of the number of entropy bits, depends both on the size of the symbol set (referred to as 'Symbol Count') and the length of the password.

Symbol Set (S)	**Symbol Count, N**	**Entropy per Symbol $(H/L)_S$, # bits**
Arabic Numerals (0-9)	10	3.3219
Hexadecimal Numerals (0-9, A-F)	16	4.0000
Case insensitive alphabet (a-z or A-Z)	26	4.7004
Case insensitive alpha numeric (a-Z or A-Z, 0-9)	36	5.1699
Case sensitive aplhabet (a-z, A-Z)	52	5.7004
Case sensitive alphanumeric (a-z, A-Z, 0-9)	62	5.9542
All ASCII printable characters (incl. space)	95	6.5699

- Given a target value for the number of entropy bits for the password (H_{target}) and a measure of the number of entropy bits per symbol $(H/L)_S$ of a symbol set S, the number of symbols to be chosen from a symbol set (L_S) to form the password with the targeted strength would be:

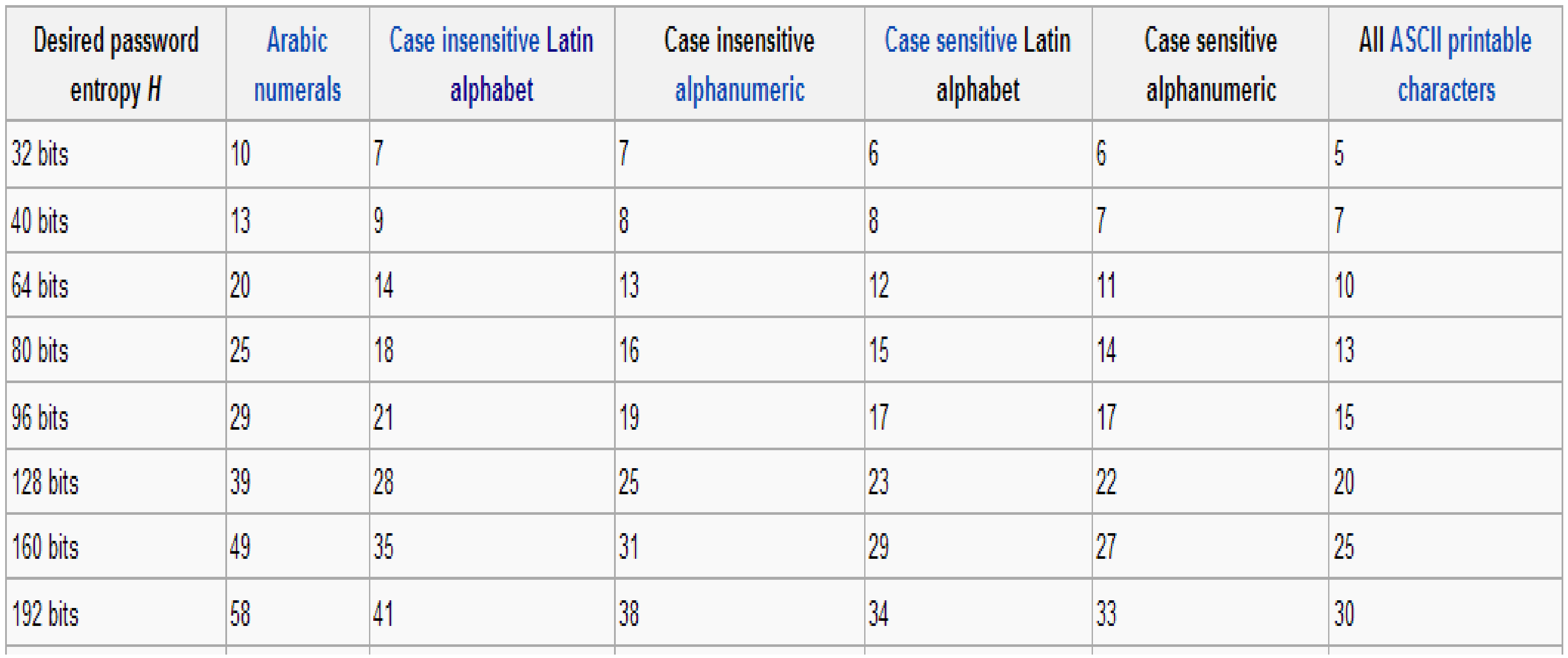
$$L_S = \left\lceil \frac{H_{target}}{(H/L)_S} \right\rceil \qquad L_S = \left\lceil \frac{H_{target}}{\log_2 N} \right\rceil$$

N – Symbol set size
L – Password length (# symbols)

Password Strength

- To form a password with a targeted value for the Number of Entropy bits, we can choose an appropriate number of symbols from any available symbol set, provided all symbols of a symbol set have an equal property of being part of the password.

Minimum lengths *L* of randomly generated passwords to achieve desired password entropy *H* for symbol sets containing *N* symbols.



Desired password entropy *H*	Arabic numerals	Case insensitive Latin alphabet	Case insensitive alphanumeric	Case sensitive Latin alphabet	Case sensitive alphanumeric	All ASCII printable characters
32 bits	10	7	7	6	6	5
40 bits	13	9	8	8	7	7
64 bits	20	14	13	12	11	10
80 bits	25	18	16	15	14	13
96 bits	29	21	19	17	17	15
128 bits	39	28	25	23	22	20
160 bits	49	35	31	29	27	25
192 bits	58	41	38	34	33	30

Source: Wikipedia: http://en.wikipedia.org/wiki/Password_strength

Use of Salt to Mitigate Dictionary Attacks

- Motivation: The hash value for a particular password will be the same each time the password is passed as the only input to the cryptographic hashing algorithm.
- With salting, an additional input, a non-secret value (say *s*), could be passed along with the password *p* as the input to the hashing algorithm ***h***(*p*, *s*).
  - Even if two users choose the same password, if their salt values are different, their hashed password values will be different too.
- The tuple <*s*, ***h***(*p*, *s*)> is stored in the password file.
- If the salt value is publicly displayed in the password file, then an attacker has to compute N hash values for every word in the dictionary, where N is the number of users listed in the password file and the N values of the salt for each word in the dictionary will be those of these N users.
- If the salt values could be hidden from the user's view,
  - If the salt is represented using N_s bits, then for every word in the dictionary, the attacker has to now compute 2^{N_s} hash values, one for each possible value of the salt in the range [0,…, 2^{Ns}-1].
- Considering the above, it also becomes very difficult to find out whether a person with passwords on two or more systems has used the same password on all of them [Each system could have its own salt value].

Math Problem on Dictionary Attacks

- (a) If a UNIX system publicly displays the 12-bit salt values of each of its 2^{10} users along with the hash values of the 8-character long passwords, compute the average number of attempts needed for an attacker to launch a dictionary attack. Assume the cardinality of the character set of the passwords is 64 and the size of the dictionary of common passwords is 2^{20}. Also, assume that there is a 25% chance that a user password is chosen from the dictionary.
- (b) If the UNIX system, described in (a), does not publicly display the salt values, compute the compute the average number of attempts needed for an attacker to launch a dictionary attack.

Math Problem on Dictionary Attacks

(a) Salt values displayed publicly in the password file

If all passwords are from the dictionary.

Average # attempts needed = $(2^{10}) * (2^{20}) / 2$

$= 2^{29}$.

If no password is from the dictionary.

Average # attempts needed $= (2^{10}) * (64^{8}) / 2$

$= (2^{10}) * ((2^{6})^{8}) / 2$

$= (2^{10}) * (2^{48}) / 2$

$= 2^{57}$.

There is a 25% chance that a password can be from the dictionary. Hence, the average number of attempts is:

$0.25*2^{29} + 0.75*2^{57} = 2^{29}*(0.25 + 0.75*2^{28}) = 0.75 *2^{29}*2^{28}$.

$= 0.75*2^{57}$.

Math Problem on Dictionary Attacks

<u>(b) Salt values NOT displayed publicly in password file</u>

If all passwords are from the dictionary.

Average # attempts needed = $(2^{12}) * (2^{20}) / 2$
$= 2^{31}$.

If no password is from the dictionary.

Average # attempts needed $= (2^{12}) * (64^{8}) / 2$
$= (2^{12}) * ((2^{6})^{8}) / 2$
$= (2^{12}) * (2^{48}) / 2$
$= 2^{59}$.

There is a 25% chance that a password can be from the dictionary. Hence, the average number of attempts is:

$0.25*2^{31} + 0.75*2^{59} = 2^{31}*(0.25 + 0.75*2^{28}) = 0.75 *2^{31}*2^{28}$.
$= 0.75*2^{59}$.

Proactive Password Cracking using Bloom Filter

- Proactive Password Cracking: Store a list of bad passwords; When a user (re)sets his password, check if it is in the bad list. If so, reject the password; otherwise, accept.
- Bloom Filter: Data structure to capture the list of bad passwords.
  - A Bloom Filter of order *k* consists of a set of *k* independent hash functions $H_1(x)$, $H_2(x)$, …, $H_k(x)$, where each hash function maps a password *x* into a value in the range 0 to N-1.

$$H_i(X_j) = y \qquad 1 \le i \le k; \qquad 1 \le j \le D; \qquad 0 \le y \le N-1$$

where

X_j = jth word in password dictionary

D = number of words in password dictionary

Bloom Filter: Procedure and Analysis

- The Bloom Filter is a hash table.
- The hash table is of size *N* bits, with all the bits initially set to 0.
- For each password, its *k* hash values are calculated, and the corresponding bits in the hash table are set to 1. If the bit already has the value 1, it remains at 1.
- When a new password is presented to the checker, its *k* hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected (considered to be in the list of bad passwords).
- Note that there cannot be false negatives (i.e., a user entered password that is in the bad list has to have all its *k* hash values index in the Bloom Filter to bit positions that are set to 1).
- However, there can be false positives (i.e., a user entered password that is not in the bad list could still have its *k* hash values that index to the Bloom Filter to bit positions that are set to 1).

$H_1(\text{undertaker}) = 25$	$H_1(\text{hulkhogan}) = 83$	$H_1(\text{xG\%\#jj98}) = 665$
$H_2(\text{undertaker}) = 998$	$H_2(\text{hulkhogan}) = 665$	$H_2(\text{xG\%\#jj98}) = 998$

Test password

Bloom Filter Analysis

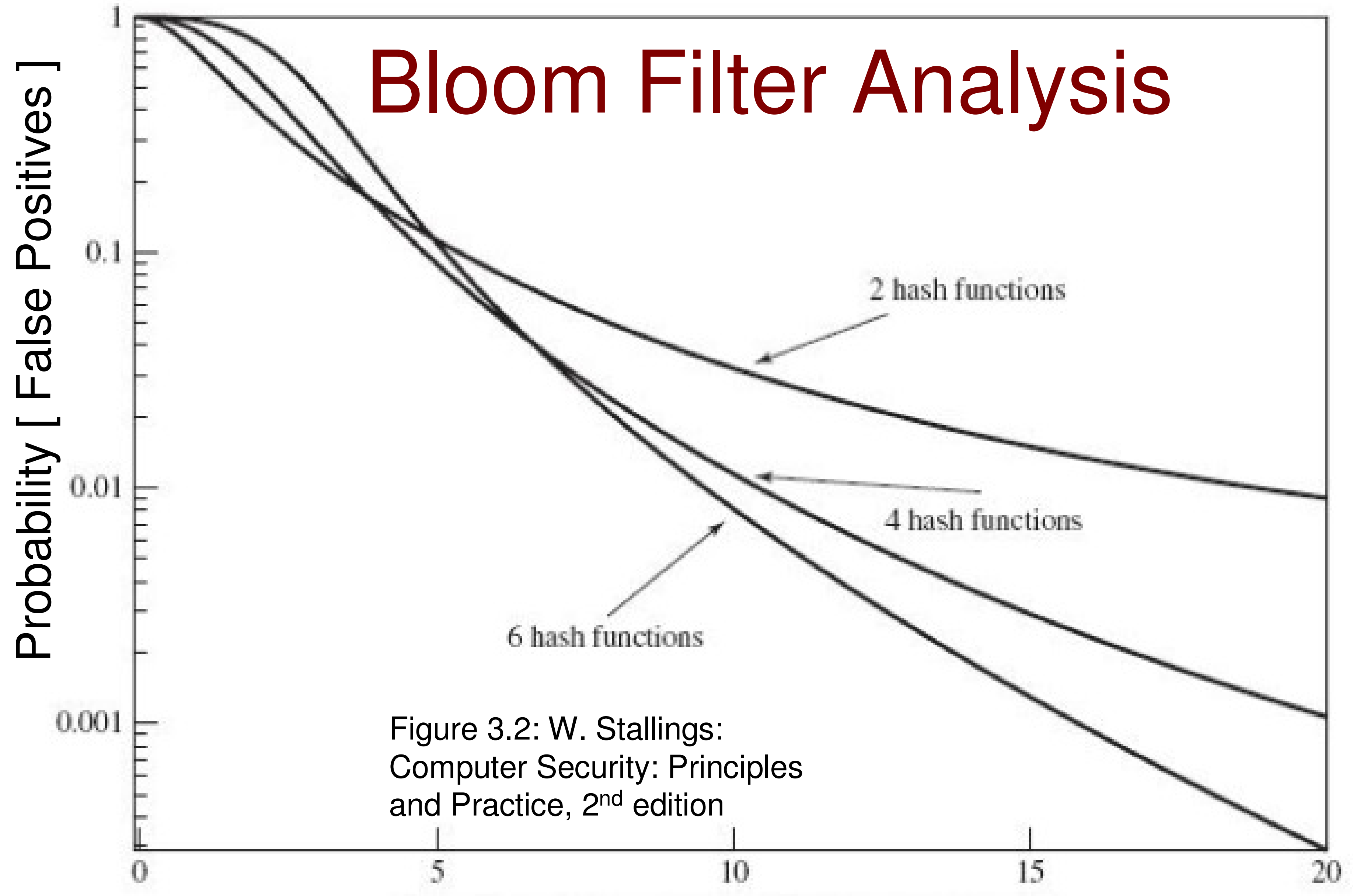


Figure 3.2: W. Stallings: Computer Security: Principles and Practice, 2nd edition

Ratio R = Max. Value in the hash table / # words in the dictionary

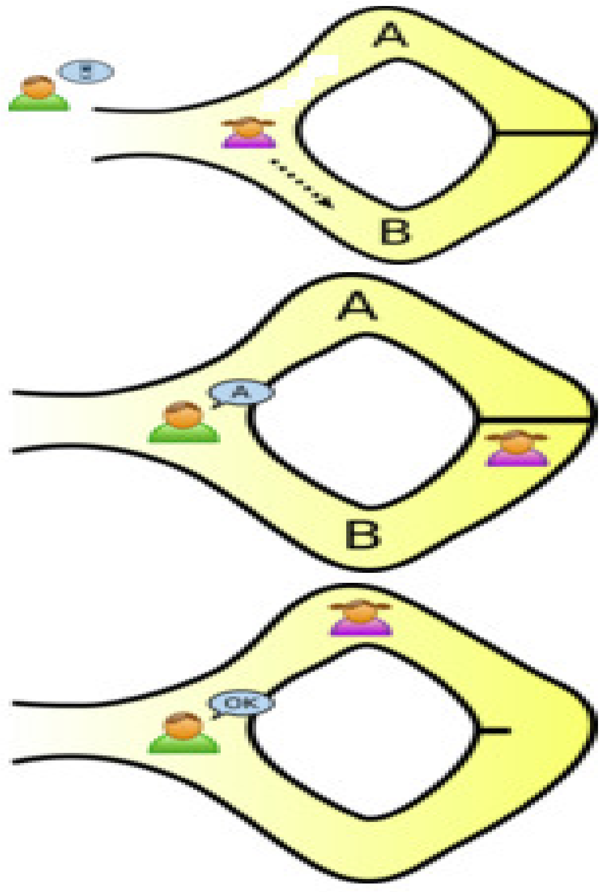
ZKPP: Challenge-Response Systems

- Zero-Knowledge Password Proof: It is used in authentication systems where one party wants to prove its identity to a second party using a password but doesn't want the second party or anybody else to learn anything about the password.
  - This technique does not even again require the system from storing the user password!! (The system should be just convinced that the user permitted to access it is the legitimate user who knows the correct password, but the system need not store the password any where in its memory)
  - This technique is based on the Zero-Knowledge Proof Protocol (ZKPP) – an interactive method (with multiple trials) for one party to prove to another that a statement is true in all the trials, without revealing anything other than the veracity of the statement.
- Tradeoff: There is always a tradeoff between the time a user wants to spend being authenticated and the level of security.

- Simple Example for Zero-Knowledge Proof Protocol (ZKPP)
  - Let there be two persons Peggy (the prover of the statement) and Victor (the verifier of the statement).
  - Peggy knows the secret word to open a magic door in a cave.
  - Victor does not know the secret word, but needs to make sure that Peggy knows the secret word before letting her access to enter the cave.
  - Peggy needs to demonstrate Victor that she knows the secret word, without telling it to Victor.

Zero-Knowledge Proof Protocol

- Let the cave be shaped like a circle, with the entrance in one side and the magic door blocking the opposite side.
- Victor waits outside the cave as Peggy goes in.
- Let the paths from the left and right of the entrance be labeled A and B.
- Peggy randomly takes either path A or B.
- Peggy knows the magic word, so she opens the door and stands inside waiting for Victor to call her back.
- Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random.
- Only if Peggy has opened the magic door, she can return on the path (including the path that she did not use to enter the cave) asked to return by Victor; otherwise she has to return through the same path that she entered.
- If Peggy did not know the magic word and she was just lucky enough to return through the same path that Victor wanted her to return, the probability of this happening per trial is ½.
- We repeat the above process over n trials. The probability that Peggy could fool Victor in all these n trials is only $(1/2)^n$.



n	$(1/2)^n$
5	0.03125
10	0.00098
15	0.000031

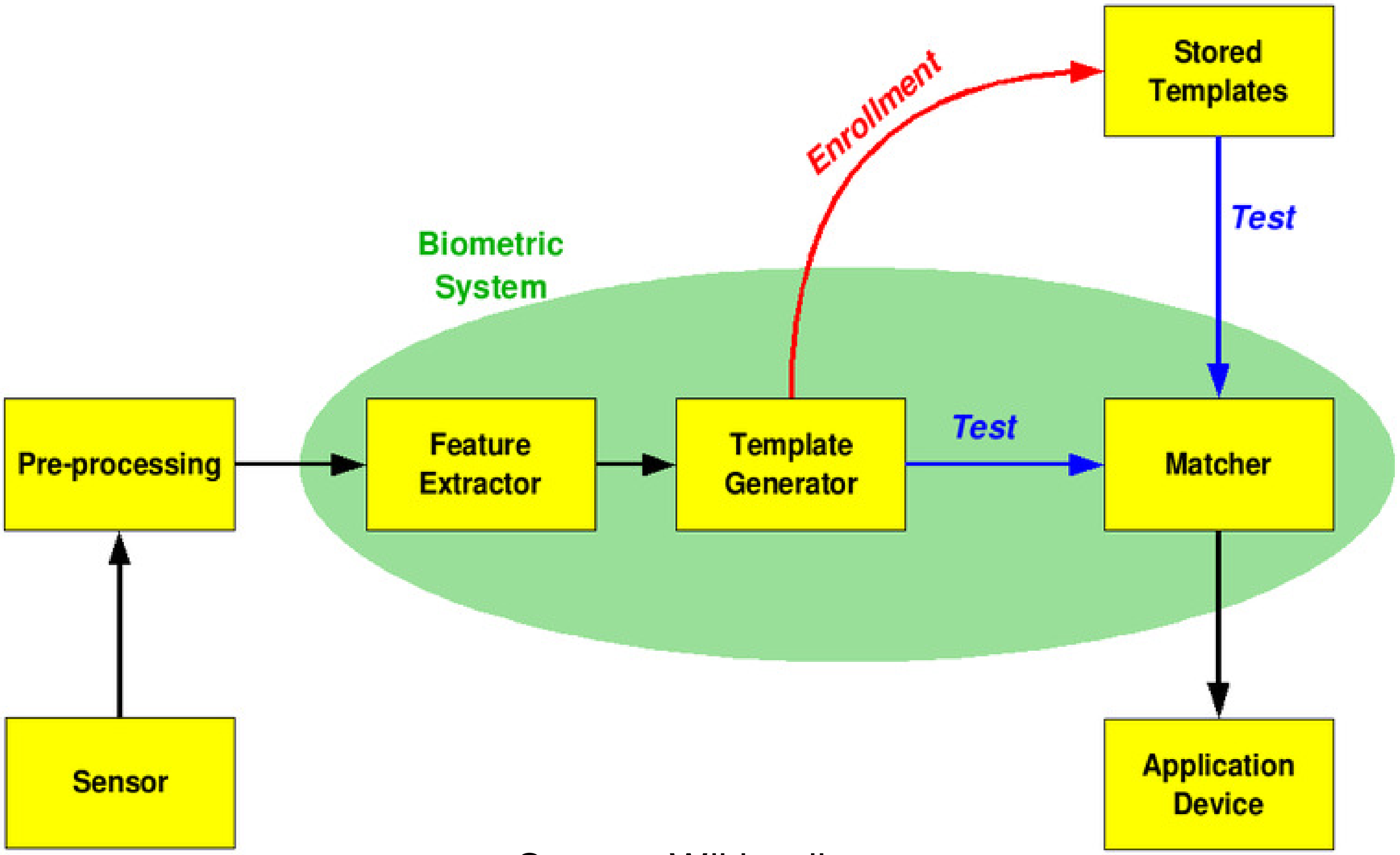
Token-based Authentication

- A token is an object that a user possesses for the purpose of user authentication.
- <u>Traditional token:</u> Memory card (magnetic stripe: containing the user ID) swiped by the user to a card reader. The card reader communicates with the server to authenticate the user.
  - Disadvantage: Anyone who possess the token can get authenticated.
- <u>Smart card:</u> Add intelligence to the token
  - Include a microprocessor, human-token interface or electronic interface to communicate with a card reader, authentication protocol
  - The user enters a PIN; the card reader transmits the user ID and PIN # to the server
- <u>Example for multi-factor authentication</u>
    - Swipe the card; enter the PIN; also have your fingerprint scanned.
    - Like ZKPP; the probability of someone forging all the three is less

Introduction to Biometrics

- Biometrics: Comprises of methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits or identifiers
  - Used to authenticate users and grant or deny access control rights to data and system resources.
- Biometric identifiers can be divided into two main classes:
  - Physiological: related to the body – often unique and can be used for identification as well as verification
    - Examples: Fingerprint, Face recognition, DNA, Palm print, Iris recognition and etc.
  - Behavioral: related to the behavior of a person – may not be unique for each person and can be used mainly for verification
    - Examples: Typing rhythm, body mechanics (gait), voice and etc.

Basic Block Diagram of a Biometric System



Source: Wikipedia

Two Modes of Biometric Systems

- Identification Mode: (primary means/source of authentication)
  - Uses biometric traits that cannot be easily forged and are supposedly unique for each user
  - Could be used for **one to many** comparison (if username is unknown) and subsequent authentication
  - Low false accept and low false error rates
  - More accurate, difficulty associated with data collection and usage
  - Examples: Fingerprint systems, Iris recognition systems, Retinal scans
- Verification Mode: (secondary source of authentication)
  - Uses biometric traits that need not be unique for each user and will incur high false accept and false error rates if used for identification; but, can be used to validate whether a user is whom he/she claims
  - Could be used for **one to one** comparison
  - Favored for ease associated with data collection and usage
  - Examples: Face recognition systems (only biometric system used for Mass Surveillance), Signature recognition systems, Voice recognition systems

2.2 Access Control

- Access control refers to preventing unauthorized access to a computer system or network.
- Access is the ability of a subject (such as an individual or a process running on a computer system) to interact with an object (such as a file or hardware device).
- Once the individual has verified their identity (authentication), access controls regulate what the individual can actually do on the system. In other words, just because a person is granted entry to the system, it does not mean that they should have access to all data the system contains.
- Access Control Models (tell what needs to be protected from what access):
  - Discretionary Access Control
  - Mandatory Access Control
  - Role-based Access Control
- Access control Mechanisms (tell how to protect the objects):
  - Access Control Matrix, Access Control List, Capability List

Discretionary Access Control (DAC)

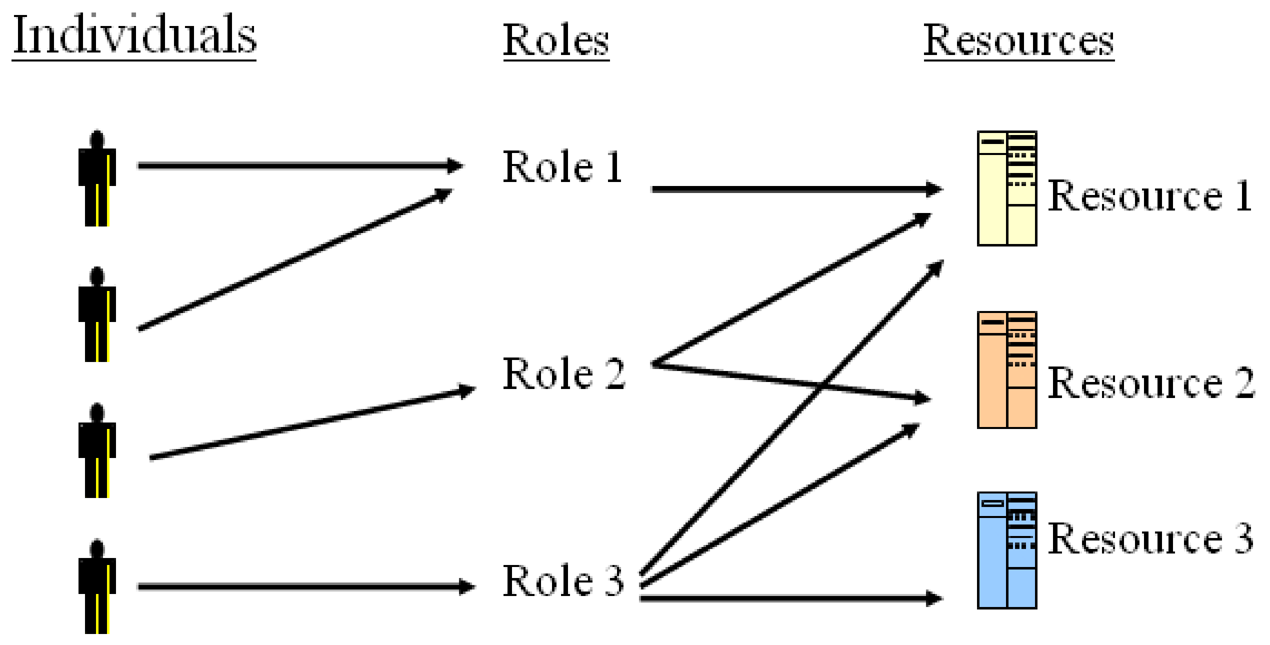
- In systems that employ discretionary access controls, the owner of an object can decide which other subjects may have access to the object and what specific access they may have.
- One common method to accomplish this is via the permission bits used in UNIX-based systems.
  - The owner of a fie can specify what permissions (read/write/execute) members in the same group may have and also what permissions all others may have.

Mandatory Access Control (MAC)

- Used in multi-level security systems.
- Access permissions are decided by the operating system and not by the subject (i.e., owner of an object).
- Each subject as well as object is identified with a security label.
- The Bell-LaPadula Confidentiality model and the Biba Integrity model are the commonly used mandatory access control models.
  - Bell-LaPadula model
    - The “no-read-up” rule states that no subject (such as a user or a program) can read information from an object (such as a file) with a security classification higher than that possessed by the subject itself.
    - The “no-write-down” rule states that a subject can write to an object only if the subject’s security classification is lower than or equal to the object’s security classification.
  - The Biba Integrity model
    - No read down rule and No write up rule

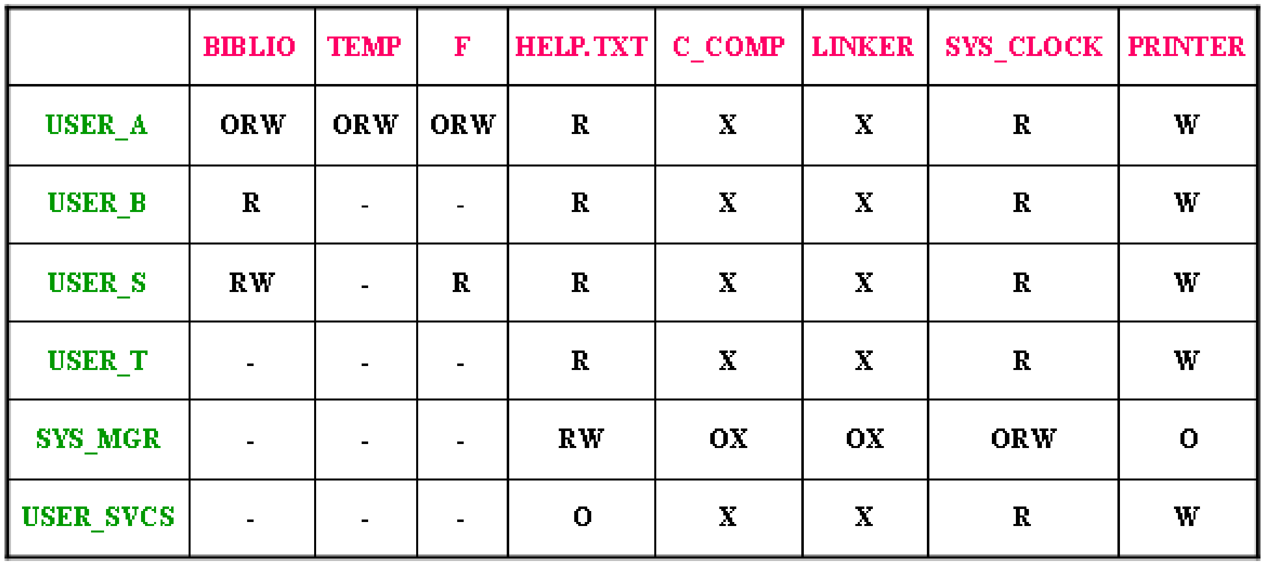
Role-Based Access Control (RBAC)

- A user has access to an object based on his/ her assigned role in the system.
- Roles are defined based on job functions.
- Permissions are defined on job authority and responsibilities of the job.
- Operations on the object are invoked based on the permissions.
- RBAC is used in Database Management Systems, Security Management and Network Operating Systems



Access Control Matrix (ACM)

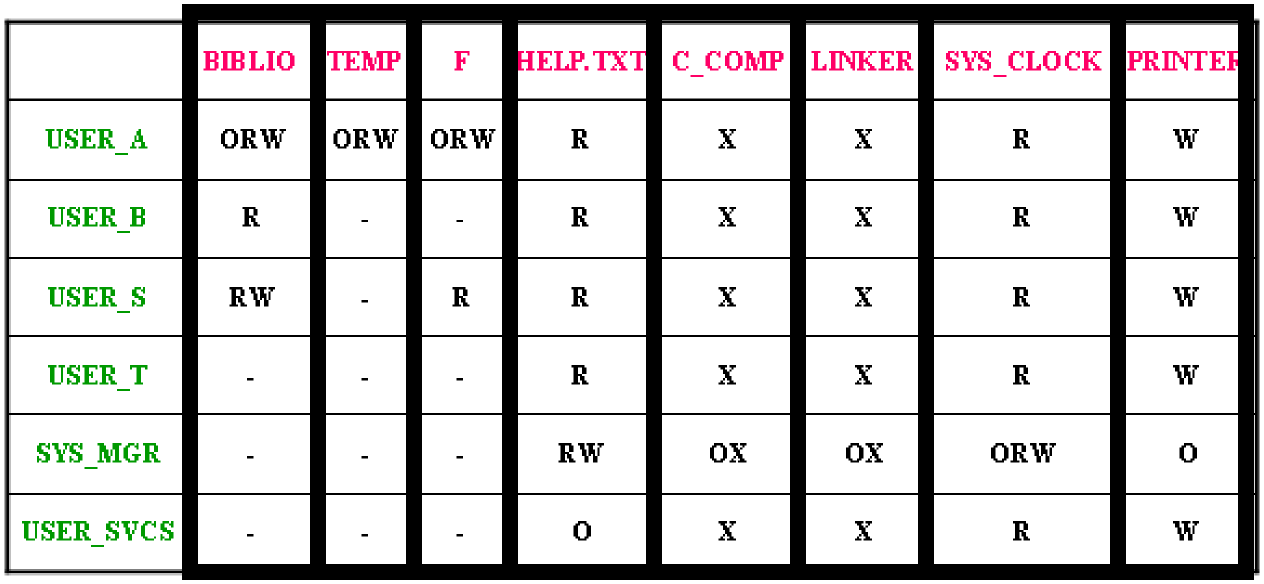
- A matrix in which each row represents a single user and each column represents a single object.
- This is a common method used in database management systems.
- Inefficient use of memory and time overhead to access the entries:
  - The matrix is huge, but is often a sparse matrix.



	BIBLIO	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER_A	ORW	ORW	ORW	R	X	X	R	W
USER_B	R	-	-	R	X	X	R	W
USER_S	RW	-	R	R	X	X	R	W
USER_T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Access Control List (ACL)

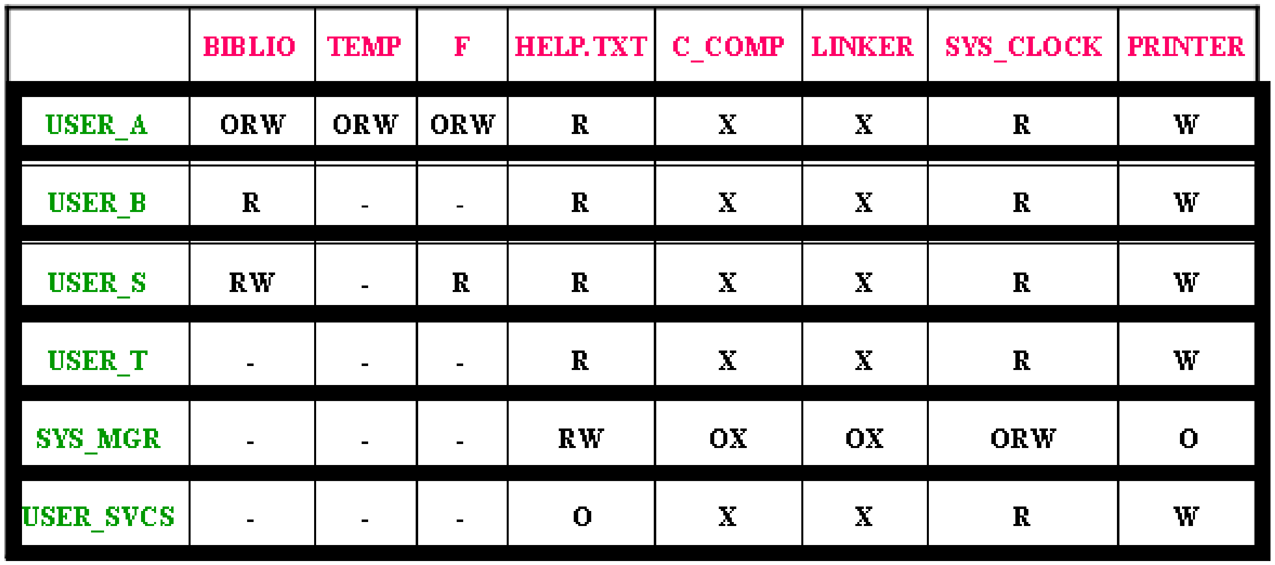
- Each column in the Access Control Matrix forms the Access Control List (ACL), one list for each object.
- Example: Biblio → (User_A, ORW) → (User_B, R) → (User_S, RW)
  - TEMP → (User_A, ORW)
  - F → (User_A, ORW) → (User_S, R)
- Each entry in the ACL for an object is called an Access Control Entry (ACE) that defines the access permissions that a specific user has on an object. Each user needs to be associated with an ACE for the object.



	BIBLIO	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER_A	ORW	ORW	ORW	R	X	X	R	W
USER_B	R	-	-	R	X	X	R	W
USER_S	RW	-	R	R	X	X	R	W
USER_T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Capability List (C-List)

- Each row in the Access Control Matrix forms the Capability List (C-List), one list for each user/process.
- Example: User_T → (Help.TXT, R) → (C_COMP, X) → ….
- C-Lists can be easily delegated when a user executes a process or when a new process is spawned by another process being executed by the user.



	BIBLIO	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER_A	ORW	ORW	ORW	R	X	X	R	W
USER_B	R	-	-	R	X	X	R	W
USER_S	RW	-	R	R	X	X	R	W
USER_T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Confused Deputy Problem

- Consider a system with 2 subjects (a user *Alice* and a user process *Compiler*) and two objects (user process *Compiler*, a file named *BILL*).
- The Access Control Matrix is as shown.
- The purpose of the Compiler is to compile the source code passed and write the debugging information to a file passed as an input argument.
- Alice invokes (has execute permission) the Compiler and passes the name of the file BILL to write the debugging information.
- If the Compiler does not check whether Alice has ‘Write’ permissions to the BILL file and goes ahead to write the debugging information to the BILL because the Compiler process itself has the ‘Write’ permission to the BILL file, then Alice would have successfully written to a file for which she does not have write permission.

	Compiler	BILL
Alice	x	---
Compiler	rx	rw

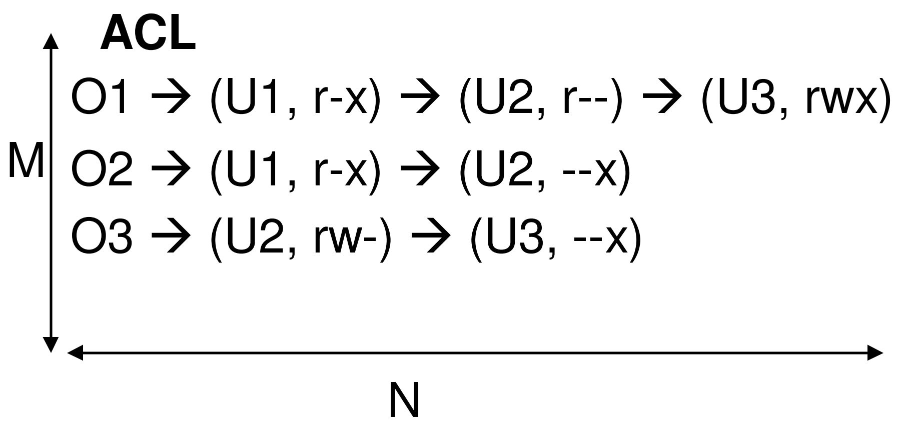
Confused Deputy Problem

- The Compiler process is a confused deputy of Alice, because even though the Compiler process has Write permissions to the BILL file, it does not know whether Alice has Write permissions to the file and it will mess up the security of the system if it executes the instructions of Alice without checking for her access permissions with the OS.
  - In other words, it has to check whether the user Alice has the appropriate access permissions to the resource (the BILL file).
- If the OS implements access control via ACLs, the breach of security, as described in the previous slide, will happen, unless the Compiler process requests the OS to validate whether Alice has ‘Write’ permission to BILL. Note that Alice does not have an ACE for BILL.
  - In other words, if an OS implements access controls through ACLs, it has to still associate each user with the objects in the form of ACEs and has to do the validation checks for every object access request from a user.
- If the OS implements access control via C-Lists, the C-List can be passed on by the user Alice to the Compiler process and the Compiler process by itself can validate whether or not Alice has access to the BILL file and need not go through the OS.
  - Note that the C-Lists are created by the OS and are read-only for all users and processes.

Time Complexity-Comparison of ACM, ACL and C-List

- Let there be N subjects (users, processes) and M objects.

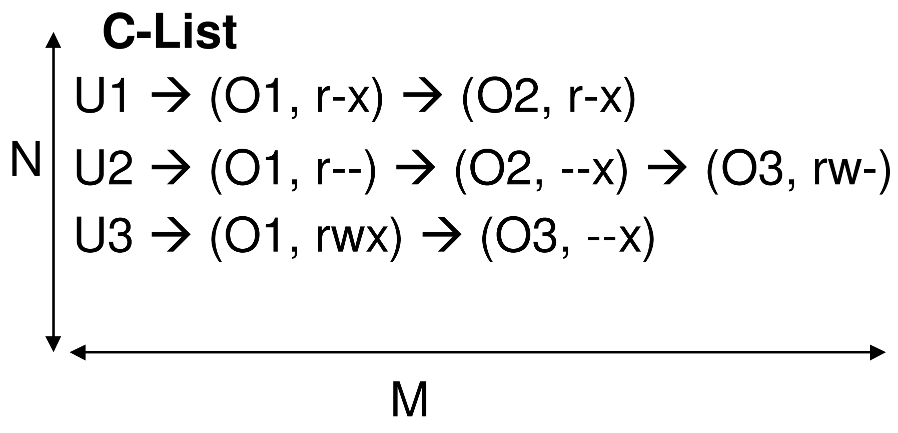
Access Control Mechanism	Ease of determining or changing authorized access during execution for a subject to an object	Ease of adding access permissions for a new subject to the different objects	Ease of creating a new object to which all subjects by default have access
ACL	O(N)	O(NM)	O(N)
C-List	O(M)	O(M)	O(NM)
ACM	O(NM)	O(NM)	O(NM)


ACL

O1 → (U1, r-x) → (U2, r--) → (U3, rwx)

O2 → (U1, r-x) → (U2, --x)

O3 → (U2, rw-) → (U3, --x)


C-List

U1 → (O1, r-x) → (O2, r-x)

U2 → (O1, r--) → (O2, --x) → (O3, rw-)

U3 → (O1, rwx) → (O3, --x)

2.3 File Protection Mechanisms

File System

- The file system is an abstraction of how the external, non-volatile memory of the computer is organized.
- Operating systems typically organize files hierarchically into folders, also called directories.
- Each resource on disk, including both data files and programs, has a set of permissions associated with it.
- File permissions (typically stored in the metadata of the file along with attributes such as type of the file, etc) are checked by the OS to determine if a file is readable, writable, or executable by a user, group of users, or a process.

UNIX File Access Control

- Associated with each file is a set of 12 protection bits.
- Nine of the protection bits specify read, write and execute permission for the owner of the file, other members of the group to which the file belongs, and all other users.
  - These form a hierarchy of owner, group and all others, with the highest relevant set of permissions being used.
- When applied to a directory,
  - the read bit lets one to list the contents of the directory
  - the write bit lets one to create/delete/rename files in the directory
  - the execute bit grants right to descend into the directory or search it for a filename.
- SetUID (Set User ID) bit and SetGID (Set Group ID) bits:
  - If these are set on an executable file: When a user (with execute privileges for this file) executes the file, the system temporarily allocates the rights of the user’s ID of the file’s creator and the file’s group, to those of the user executing the file (referred to as the effective user id and effective group id at the time of execution, in addition to the real user id and real group id).

UNIX File Access Control

- SetUID (Set User ID) bit and SetGID (Set Group ID) bits:
  - If these are set on an executable file:
    - This change is effective only while the program is being executed.
    - This feature enables the creation and use of privileged programs that may use files normally inaccessible to other users (e.g., passwd program)
  - If these are set on a directory, the SetGID permission indicates that newly created files will inherit the group of this directory. The SetUID permission is ignored.

- Sticky bit
  - When set for a file, the file contents are not totally moved out of main memory after execution, and are stored in swap space: set for frequently used programs to speedup execution.
  - When set for a directory, only the owner of a file in the directory can move, rename or delete the file: useful for managing files in shared temporary directories.

UNIX File Permissions

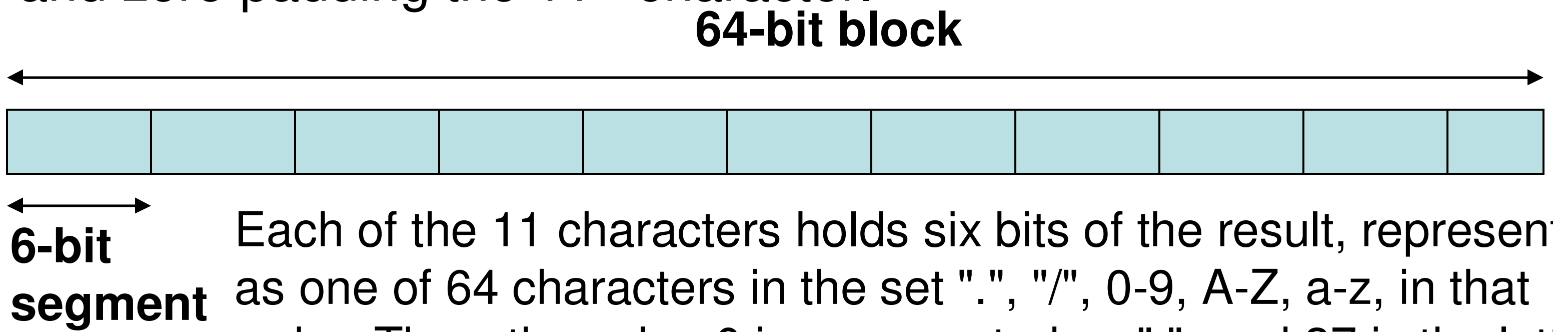
- The owner uses the `chmod` command to set the access rights of a file and can use the `chown` command to change the owner or group of a file.
- The Access rights are: Read (r – 4), Write (w – 2) and Execute (x – 1), nothing (0).
- Each file has associated permissions of the form

$$\begin{array}{ccc} \textbf{rwx} & \textbf{rwx} & \textbf{rwx} \\ \longleftrightarrow & \longleftrightarrow & \longleftrightarrow \\ \textbf{owner} & \textbf{group} & \textbf{others} \end{array}$$

- If a file has to be given more than one access right to a class, we have to add their corresponding values.
- **Examples:**
  - Consider a file A.txt. To set read, write and execute permissions to its owner, read and execute only permission to the group and read-only permission to others, use the chmod command as **chmod 754 A.txt**
  - To set the SetUID bit (4) and SetGID bit (2) to a file and set read, write and execute permissions to the owner, read/write permission to group and read permissions to others, use the chmod command as: **chmod 6764 A.txt**
  - To set the sticky bit to a file, use: **chmod +t A.txt**

Standard UNIX Password Encryption

- The first 8 ASCII characters of a user’s password are used. If your password is less than 8 characters in length, then 0 bits are padded to make it 8*7 = 56 bits in length.
- The 56 bits of the user’s password is used as the DES key. A constant 64-bit block (consisting of all zero bits) is then encrypted via DES 25 times (the result of each encryption being used to feed the next round), using the 56-bit user password as the key.
- A 12-bit salt value drives the DES P- and S-box tables used.
- The resultant 64-bits is converted into a string of 11 printable ASCII characters, by encoding every six bits into a printable ASCII character and zero padding the 11th character.



Each of the 11 characters holds six bits of the result, represented as one of 64 characters in the set ".", "/", 0-9, A-Z, a-z, in that order. Thus, the value 0 is represented as ".", and 37 is the letter “Z"

.	**/**	**0**	**1**	**…..**	**9**	**A**	**B**	**…..**	**Z**	**a**	**b**	**….**	**Z**
0	**1**	**2**	**3**		**11**	**12**	**13**		**37**	**38**	**39**		**63**

UNIX System Password File

- Can an encrypted system password file (with user name and encrypted password) be made public?
  - What happens if there are two users who choose the same password. The encrypted password appearing in the password file would also be the same. So a user can realize that the other user is most likely to have chosen the same password.
- Solution: Use Salting
  - Generate a 12-bit salt, (which is normally obtained by dividing the system clock time by 4096 and the remainder is used as a salt) at the time of setting the password.
  - Use the 12-bit salt to decide the Expansion table to be used for encrypting the 64-bit 0 block and the subsequent encrypted strings.
  - A given user password can now be encrypted to 4096 different ciphertexts.
  - So, even if two encrypted passwords are the same, their source could be different from each other.
  - Salting is used to reduce the possibility of a dictionary attack in which the attacker would have a mapping between the passwords in plaintexts and their encrypted versions. Instead of having a one-to-one mapping between a plaintext string and its encrypted version, an attacker now needs to have a one-to-4096 mapping between a plaintext string and its encrypted versions.

UNIX System Password File

- On any UNIX-like file system, the user identity information is stored in the “passwd” file and it is located in the “/etc/” directory.
- The “passwd” file has the following format: 7 colon-delimited fields and the fields are in the following order:
  - Username, encrypted password along with the salt, user ID, group ID, full name, Home directory, Shell

  - User ID – is a numeric identifier, which the OS uses to identify which files belong to the user. The system always thinks of the user in terms of a number. It uses the “passwd” file to convert the number into a more human-friendly form, the “username”. The “username” is the name assigned by the system administrator and will be used to log in to the system.
  - Group ID – a UNIX group may contain none, one or more users, who will be able to access the files and directories owned by that group, based on that group’s permissions. This is useful for sharing files between two people, as a file can have only one owner.
    - User Private Groups – each user is assigned their own group, identified by their username. The user is the only member of the group.
    - User private groups are used in most modern day implementations
  - Home directory – location where all the user files are usually stored
  - Shell – the command line that provides the user interface to the UNIX OS

UNIX System Password File

- The encrypted password + salt field is a 13-character field: the first two characters are the salt and the next 11 characters form the encrypted password.

```
rachel:eH5/.mj7NB3dx:181:100:Rachel Cohen:/u/rachel:/bin/ksh
arlin:f8fk3j1OIf34.:182:100:Arlin Steinberg:/u/arlin:/bin/csh
```

- In the above example, 5/.mj7NB3dx is the encrypted password and eH is the salt for username “rachel”.

- Traditional UNIX systems keep user account information, including the encrypted passwords, in the “/etc/passwd” text file.
- The “/etc/passwd” File is used by many tools (such as “ls”) to display file ownerships, etc., by matching user ID with the username field. As a result, the file needs to be world-readable and consequentially can be somewhat of a security risk.

- Solution: Store the actual encrypted password in another file called the “/etc/shadow” file and it is readable only by the root. This file also contains the password aging information along with the account information.

UNIX System Password File

- In most modern day UNIX and LINUX systems, the encrypted password is not stored in the "/etc/passwd" file and is stored in the "/etc/shadow" file.
- If the encrypted password is stored in the "/etc/shadow" file, then the password holder field in each row of the "/etc/passwd" file is filled with just character "x".

```
smithj:x:561:561:Joe Smith:/home/smithj:/bin/bash
```

- The "/etc/shadow" file contains a row for each user; each row contains 9 fields, each separated by a ":", in the form:

```
login-id:password:lastchg:min:max:warn:inactive:expire:flag
```

  – Login-id: User name
  – Password: 13 character (2 character salt + 11 character encrypted password)
  – lastchg: number of days, since the password was last changed
  – Min: the minimum number of days before password may be changed
  – Max: the maximum number of days, after which the password must be changed
  – Warn: the number of days to warn user of an expiring password
  – Inactive: the number of days the account can be inactive, without being used
  – Expire: the number of days (since Jan 1, 1970) that the account should be disabled
  – Flag – reserved for future use

Finding the Salt Value from the UNIX /etc/passwd File

```
rachel:eH5/.mj7NB3dx:181:100:Rachel Cohen:/u/rachel:/bin/ksh
```

The first two characters (of the 13 characters) in the second field represent the salt. In the above example, they are ‘eH’

.	/	0	1	.....	9	A	B	C	D	E	F	G	H	.....	Z	a	b	c	d	e	....	Z
0	**1**	**2**	**3**		**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**		**37**	**38**	**39**	**40**	**41**	**42**		**63**

The 6-bit equivalents for e and H are 42 and 19 respectively.

e						**H**					
1	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
42						**19**					

2048	1024	512	256	128	64	32	16	8	4	2	1
1	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**

12-bit Integer Salt Value

2048 + 512 +128 + 16 + 2 + 1 = 2707

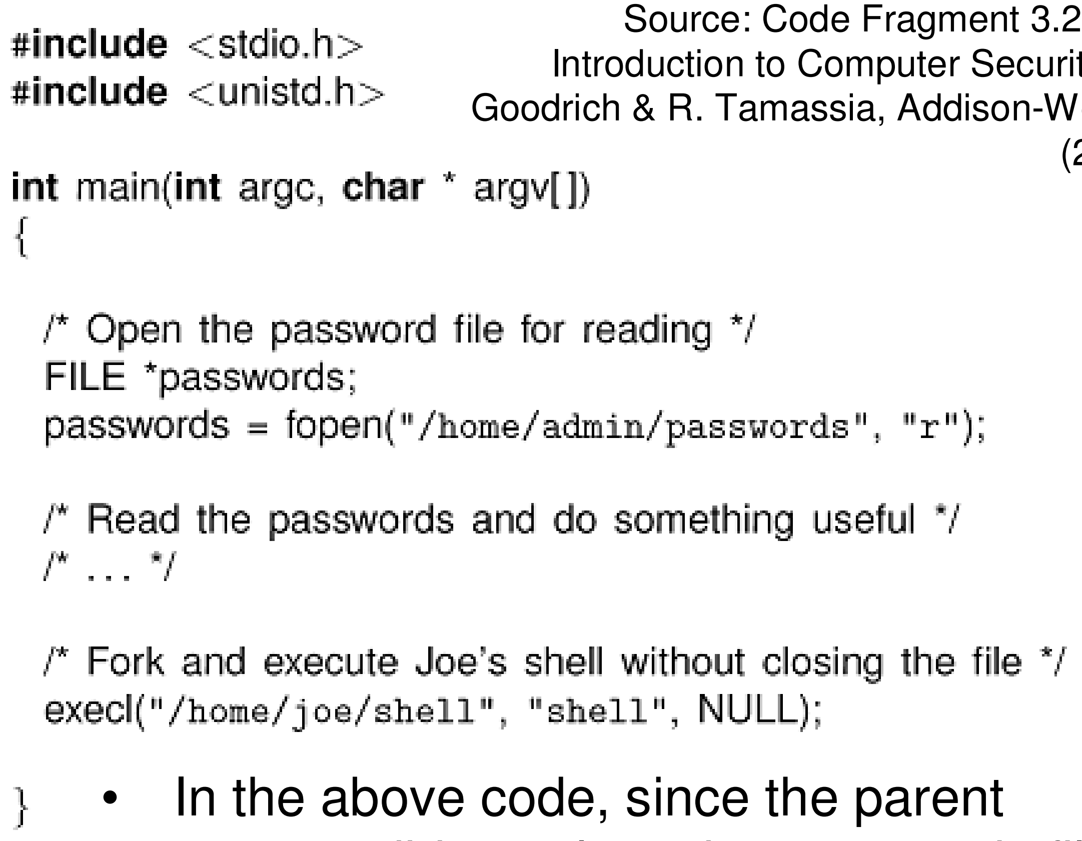
Working with File Descriptors

- A “file descriptor” is an abstract indicator used by the kernel to access the files opened and currently used by a process, instead of requiring to always send the entire path information to access the file.
- When a process attempts to open a file located at a certain path, the OS kernel assigns to the user a file descriptor for the file and stores the same in a “file descriptor table” that maps the file descriptor with the file’s location on the disk.
- Before the kernel assigns the file descriptor for the process, the kernel makes sure that the calling process has all the appropriate access permissions on the file.
- The file descriptor table cannot be directly accessed by the user process and can be accessed only through system calls (e.g., open, close).
- Before termination, the user process should make sure to return the open file descriptor by initiating a “close” system call.

File Descriptor Leak Vulnerability

- Causing Factors
- When a parent process spawns a child process, the latter inherits copies of all of the file descriptors that are open in the parent.
- The OS only checks whether a process has permissions to read or write to a file only at the time of creating a file descriptor entry and not at the time of using it to read or write to a file.

Source: Code Fragment 3.2 from Introduction to Computer Security, M. Goodrich & R. Tamassia, Addison-Wesley (2011)



```
#include <stdio.h>
#include <unistd.h>

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

  /* Open the password file for reading */
  FILE *passwords;
  passwords = fopen("/home/admin/passwords", "r");

  /* Read the passwords and do something useful */
  /* . . . */

  /* Fork and execute Joe's shell without closing the file */
  execl("/home/joe/shell", "shell", NULL);

}
```

- In the above code, since the parent process did not close the passwords file descriptor before spawning the child process, the latter can also read the file, even if the parent process' intention is not to have any child process read the password file.

Symbolic Links and Shortcuts

- Motivation: To avoid copying an entire file to different locations. Instead, a user can create a link to the file at the different locations (e.g., Desktop) from which the file will be accessed. If the user makes any change to the underlying file, all links to the file will automatically be referring to the updated version.
- Unix-based systems accomplish the above through symbolic links (created using the *ln* command) and Windows through *shortcuts*.
- In Unix, the use of symbolic links is completely transparent to the user applications. For example, if a user passes the path to a symbolic link file as the input argument to a file reader program, the OS will follow the link to the actual file pointed to by the symbolic link and makes the file available to the user application.
  - However, this could be misused by a malicious party. If a file reader program is written to read all files except the passwords file located at /home/admin/passwords, a malicious user could trick the reader program by passing a file that has a symbolic link to the password file. The trick will work if the file reader program merely checks the path/filename specified by the user.
  - To prevent such aliasing based attacks, the file reader program should not accept filenames with symbolic links. This can be validated by issuing a stat system call on the filename and checking the return value for a symbolic link.

Symbolic Links and Shortcuts

- In Windows, the shortcuts are treated like ordinary files by the file system and by software programs that are not aware of them.
- Only software programs that understand shortcuts (such as the Windows shell and file browsers) treat them as references to other files.
- This way, the aliasing attack that could be possible with Symbolic links in Unix-based systems is avoided in Windows systems.

- Unlike the Unix symbolic links, Windows shortcuts maintain the references to their targets even if the target is moved or renamed.

2.4 Firewalls and Intrusion Detection Systems

Firewalls

- Firewall is a software running on a dedicated hardware.
  - No other application is run on this hardware: to protect the firewalls rules from being tampered.

- A firewall is usually placed in the network boundary, monitoring and filtering incoming and outgoing traffic.
  - Ingress filtering (filter incoming traffic)
  - Egress filtering (filter outgoing traffic)
  - All traffic for a network has to be designed to go through a firewall

- A firewall has to be designed with a combination of black-list (default-allow) and white-list (default-deny) approaches
  - Default-allow: Allow all packets except those that match the blacklisted networks, ports, etc
  - Default-deny: Allow only those that match the preferred networks, ports, etc; drop other packets.

Packet Filter and Stateful Firewalls

- Both packet filter and stateful firewalls operate only on the packet headers and not on the data.

- Packet Filter: Stateless firewalls that operate on a per-packet basis
  - Decisions are independently taken on each packet and are not remembered (no state info)

- Stateful Firewalls: Monitors sessions and maintains state information on the packets seen for the session
  - Could be used to detect bandwidth used by a particular source and enforce users from downloading more than certain amount of bytes within a time period
  - Could drop excessive traffic coming to specific servers (prevent denial of service attacks)

Application Firewall

- The packet filter and stateful firewall look at only the packet headers. The application proxy firewall scans through the entire packet (including the application data) and makes sure if it could be forwarded in/out.
- An application firewall protecting an internal network of clients from being attacked by an external server/user is called a <u>Proxy Firewall</u>.
  - Example: An application firewall that protects an internal network of desktop/ office machines from users attempting to connect after office hours.
- An application firewall protecting an internal network of servers from being attacked by an external client is called a <u>Reverse Proxy Firewall</u>
  - Example: A reverse proxy firewall hosted to protect a sales network (comprising of various servers – database server, file server, etc) monitors every incoming packet to make sure it does not have any malicious scripts to cause any command injection attacks (XSS, XSRF or SQL-injection) or buffer overflow attacks.

Personal Firewall

- Personal firewalls are different from the network firewalls (packet filter, stateful and application firewalls)
  - A personal firewall runs on a host (that it wants to protect) on which several other applications also run
  - They are kind of like all-in-one: Could scan for virus, block traffic to/from specific sites, etc.
  - The rules of a personal firewall can be more customzied to the demands of the user, unlike the network firewalls that are configured based on only network-wide policies.
  - Each Operating System has its own variant of personal firewalls (Linux kernel firewall - iptables; Microsoft Security Essentials for Windows systems)

Intrusion Detection Systems (IDS)

- An IDS is operated inside a network to monitor for any abnormal or malicious activities; operates in promiscuous mode
  - Passive IDS: Raises an alarm and keeps quiet
  - Active IDS: Reacts to the attack traffic and tries to control the attack (Intrusion Prevention Systems)

- Network-based IDS: Monitors the network activities; runs on dedicated computers; handles large volume of traffic; coordinates with network firewalls to update the filtering rules

- Host-based IDS: Runs on individual hosts; customized for the host needs; logs all user activities; could also as anti-virus scanner; takes up host resources for monitoring
- Signature-based IDS: Goes strictly by rules (can detect only known attacks) – more false negatives and few false positives

- Anomaly-based IDS: Raises alarm when notices unusual behaviors (e.g., changing password right after login; unusual amount of traffic leaving the host): Could detect even zero-day attacks (attacks for which there exists no pre-determined signature – more false positives and few false negatives.