MONASH University
Information Technology

FIT2093 INTRODUCTION TO CYBERSECURITY

Week 8 Lecture

Software & System Security II

Entity AUTHentication & Access Control

S1 2022


Outline

Software & System Security: Entity AUTHentication & Access Control

- User AUTH: identify who a user is
  - Unique factors
    - what you **know** (**w**), / **have** (**w**), / **are** (**w**)
    - **unforgeability** vs **usability**
  - **Case study: Password-based User AUTH**
    - **Password management principles**
    - **Password management in UNIX**
  - **Case study: Biometrics**
- Access Control: restrict what a user can do
  - Case study: Access control in UNIX

MONASH University

User AUTH: Unique **Factors**


Unique Factors

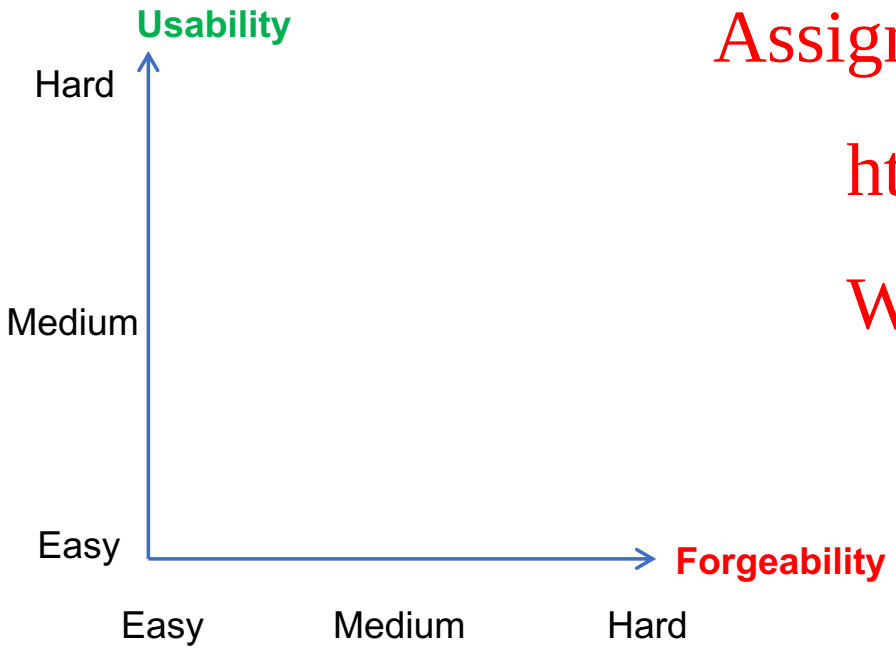
Software & System Security: Entity AUTHentication

- *Recap: AUTH = uniquely identify individual*
- The Problem:
  - impersonation
  - repudiation
- The Solution:
  - **unique** factors

Q: Give an example of a unique factor that could be used for user AUTHentication.

Q: Rate the following aspects of your AUTH factor:

- How hard it is to **forge** this AUTH factor? (Easy, Medium, Hard)
- How hard it is to **use** this AUTH factor? (Easy, Medium, Hard)




Unique Factors

Software & System Security: Entity AUTHentication

Usability: **Forgeabiity** ----------	Easy	Medium	Hard
Hard			
Medium			
Easy			

- Forgeability: ...
- Accessibilty: …

Q: which factor is best?

Unforgeability & False Sense of Security


Software & System Security: Entity AUTHentication

- Q: which factor is unbreakable?
- Q: what is the consequence of assuming too much on a factor's unforgeability?

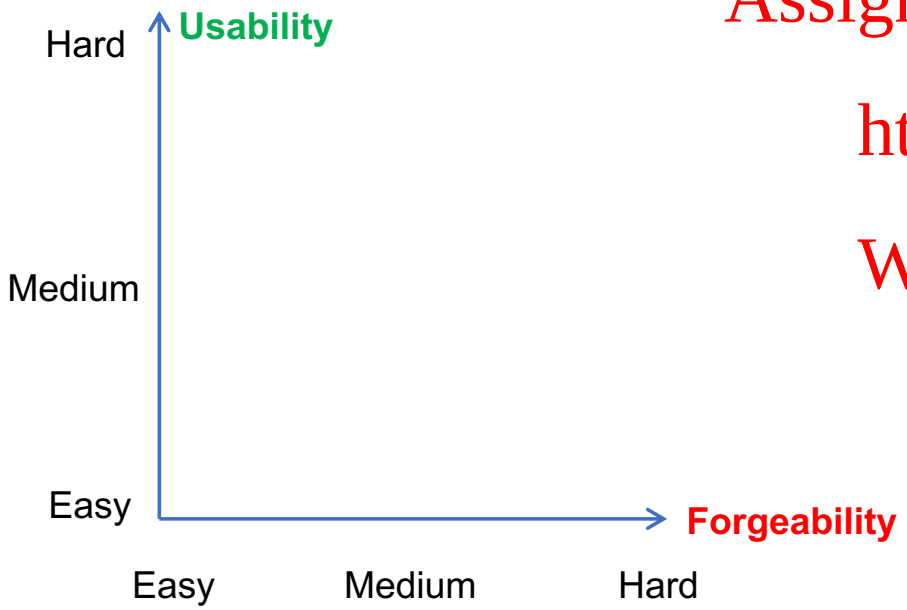
Multi-Factor AUTH

Software & System Security: Entity AUTHentication


- The Problem:
  - **no best** factor
- The Solution: multi-factor AUTH
  - have eggs in **multiple** baskets, use **multiple factors**
    - breaking each factor needs **separate** effort
    - need to break **all factors** to break system

Q: What two-factor combo is most effective?

e.g. Pwd+phone? Card+PIN? Face+Pwd? …



MONASH University

User AUTH Case Study: **Password** based AUTH


Password based AUTH

Software & System Security: Entity AUTHentication

- AUTH based on what you know
- Challenge-Response system btw Prover & Verifier
- Prover: how to prove?
  - show unique factor
  - **Q:** still unique after show?
- Verifier: how to check?
  - compare

Q: Explain an assumption that security of password-based AUTH depends on.

Security of Password based AUTH

Software & System Security: Entity AUTHentication

- Password based AUTH security depends on what assumption(s)?
  - **unique** / only you know
  - you won't **tell**
  - others can't **guess**
  - verifying server will **protect it**

- Q: how can any of these assumptions be invalidated?

Attacks on Password based AUTH


Software & System Security: Entity AUTHentication

- Breaking the Uniqueness "**Only you know**" assumption: observe
  - leakage
  - shoulder surfing
  - device hijacking
  - keystroke logging

- Breaking the "**You won't tell**" assumption: phish
  - social engineering: ignorance / nonchalance / gullibility
  - phishing / scam


Attacks on Password based AUTH

Software & System Security: Entity AUTHentication

- Breaking the "**Others can't guess**" assumption: guess
  - human laziness / ignorance / nonchalance
    - same password, different systems
    - easy to remember e.g. common word, personal info

      → so, easy to guess


Attacks on Password based AUTH

Software & System Security: Entity AUTHentication

- Breaking the “Others can’t guess” assumption
  - bruteforce guessing
    - **random secret**: infeasible
      - e.g. 64-bit secret key of lightweight encryption
      - need to try 2^{64} possible values
    - **password**: feasible
      - e.g. 8-character password
      - 95 printable ASCII characters vs $2^8 = 256$ possible values
  - bruteforce guessing passwords is a feasible attack: dictionary attack

Attacks on Password based AUTH


Software & System Security: Entity AUTHentication

- dictionary attack
  - **online**
    - need to **interact** with verifier to check if guess correct
    - e.g. PIN at ATM machine
  - **offline**
    - can check **without** needing **verifier** presence, attacker himself verifies on attacker machine
    - e.g. hashed password file
- ***Q:*** *Guessing your friend's phone screen password when s/he's not there: online or offline?*


Countermeaseasures vs Attacks

Software & System Security: Entity AUTHentication

- online dictionary attack: how to prevent?
  - limit the number of verification requests
  - auto lockout / penalty (*e.g. Bluetooth, Android screen pw*)
- offline dictionary attack: how to prevent?
  - increase the password length/randomness
    - password policies against using common passwords, to choose hard-to-guess passwords
    - ***Q****: Which is more secure: 8-char letter or number, or 9-char letter only?*


Countermeaseasures vs Attacks

Software & System Security: Entity AUTHentication

- monitoring / eavesdropping
  - *electronic*: encrypted network links (e.g. SSL/TLS)
  - *shoulder surfing*: special screens
- device hijacking / keystroke logging
  - automatic logout

MONASH University

Password-based AUTH: **Password** Management Principles


Password Management Principle: Choice

Software & System Security: Entity AUTHentication

- **choice** of password
  - dilemma: ↓ guessable passwords, ↑ memorability
  - **hard to guess**: not personal info / not common
    - user education
    - computer generated
    - proactive password checking: disallow weak ones
    - reactive password checking: crack own system, replace guessed ones
  - be wary of defaults: *e.g. weak default, same default for all*

Password Management Principle: Freshness


Software & System Security: Entity AUTHentication

- **freshness** of passwords
  - **change** regularly
  - auto password **aging**: *e.g. auto prompt to change*
  - keep password history & **prevent re-use** of earlier passwords
  - ***Q**: why prevent re-use of old passwords?*

Password Management Principle: Lifetime


Software & System Security: Entity AUTHentication

- **one-timeness** of passwords
  - challenge varies every time / so used only once
  - short time / one-off / one-time password (OTP)
    - *e.g. online banking with second factor (phone, OTP device)*
    - ***Q**: why do we need long term passwords? Why not only use OTP?*

Password Management Principle: Storage


Software & System Security: Entity AUTHentication

- Assumption: verifying server will protect password P
  - never store P in the clear
    - store transformed password f(P)
    - common operating systems (OS) use this idea, where f = **one-way hash function**

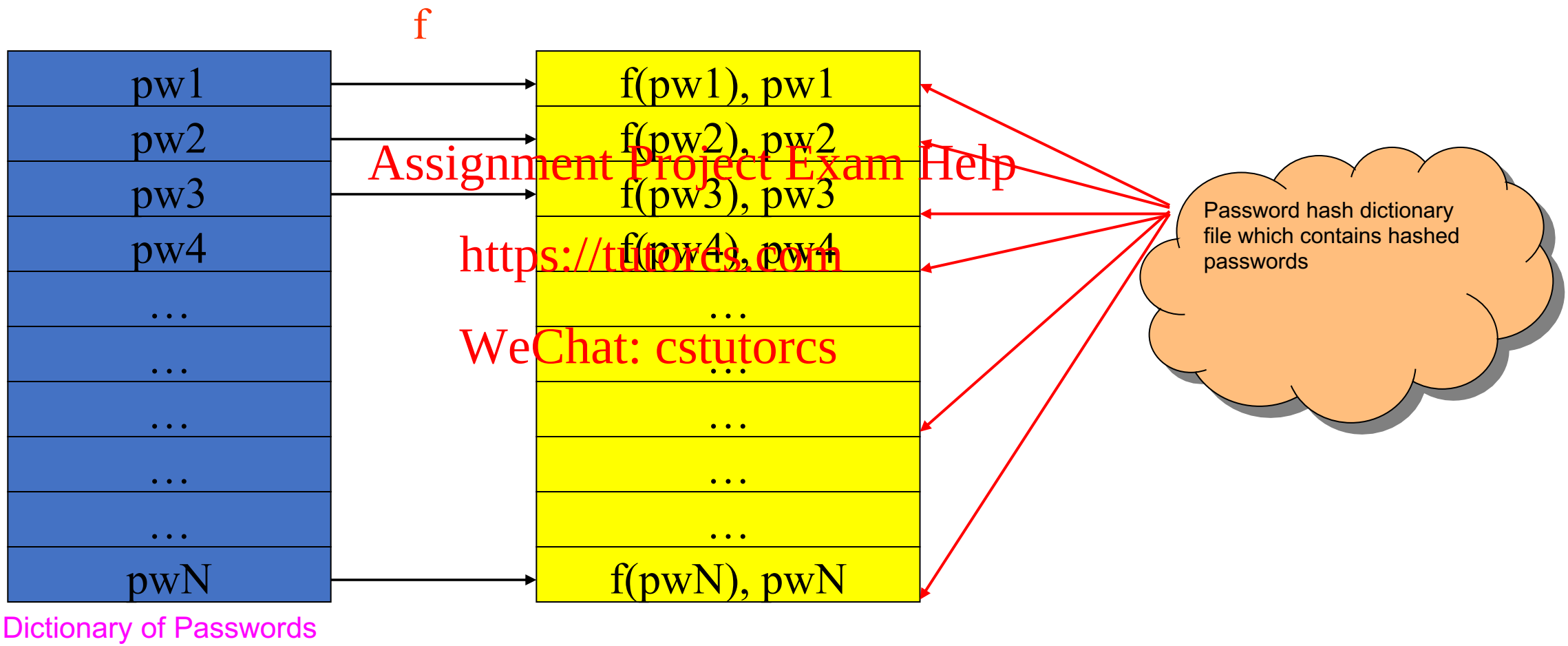
- ***Q:*** *what properties should f( ) have?*
- ***A****: Even if attacker gets f(P)* and knows f( )
  - A cannot get password P (**one-wayness**)
  - A cannot find another P' s.t. f(P') = f(P) (**one-wayness**)

Precomputation Attack on Stored Passwords


Software & System Security: Entity AUTHentication

- Limitation of password hashing: brute-force precomputation dictionary attack
- Precomputation phase: (slow)
  - create a dictionary of possible passwords P
  - for each P in dictionary, compute hash f(P)
  - store (f(P), P) pairs in table indexed by f(P) (*reverse dictionary*)
- Lookup phase: **after** hash file is exposed (very fast)
  - look up each file hash f(P) in reverse dictionary
  - Off-line attack: No need for on-line interaction for each password guess
  - can expose many system passwords very quickly

Dictionary Attack: How it works?



Defense against Precomputation Attacks

Software & System Security: Entity AUTHentication

Q: If hash function f takes 1 microsecond to hash a password: How long would it take to make a dictionary for all possible 7-letter-only passwords? What if f takes 0.1 sec to hash a password?

- Precomputation phase: **before** hash file is exposed (slow)
  - create a dictionary of possible passwords P
  - for each P in dictionary, compute hash f(P)
    - ***Defense 1 (time):*** *slow f down (special password hash functions)*
  - store (f(P), P) pairs in table indexed by f(P) (*reverse dictionary*)
    - ***Defense 2 (memory):*** *make the table too big to fit in memory (**salting**)*
- Lookup phase: **after** hash file is exposed (very fast)
  - when system password hash file f(P) exposed, look up each file hash in reverse dictionary
  - can expose many system passwords very quickly


Salting: Defense 2 against Precomputation Attacks

- Defense 2: Precomputation infeasible if **table too big** to precompute in memory
- Countermeasure:
  - append randomness ("salt") R to password P before hashing P
  - instead of storing f(P), store (**R, f(P,R)**)
  - Salt R is randomly chosen L-bit string (say L = 128 bit) chosen **independently for each password**

- Precomputation attack now may need to guess salt R in advance, or compute 2^L tables (one for every possible R value)
  - precomputation attack infeasible if L large enough
    - E.g. memory of 1 TB = 2^{40} bytes << 2^{128} bytes

Summary: Use of one-way hashing + Salting on Passwords


Software & System Security: Entity AUTHentication

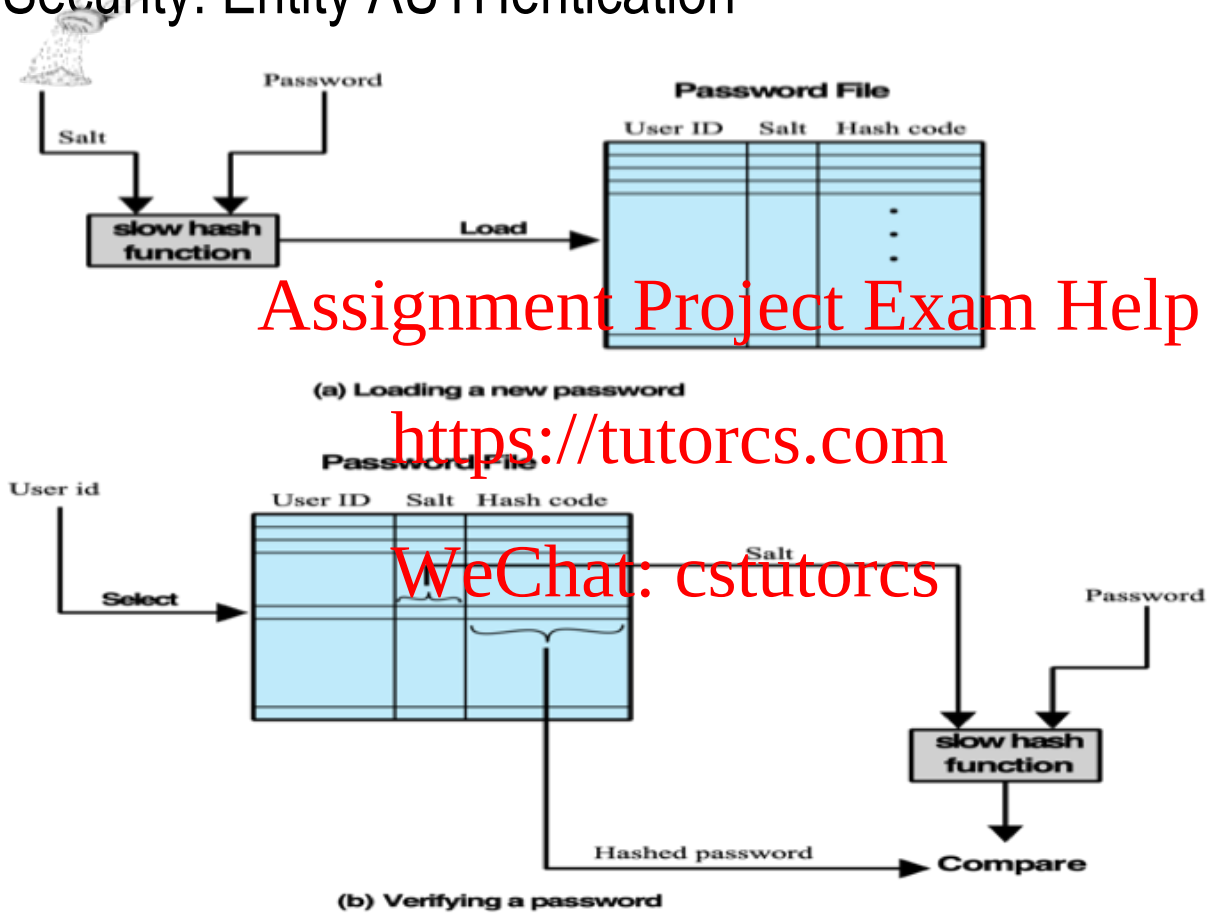
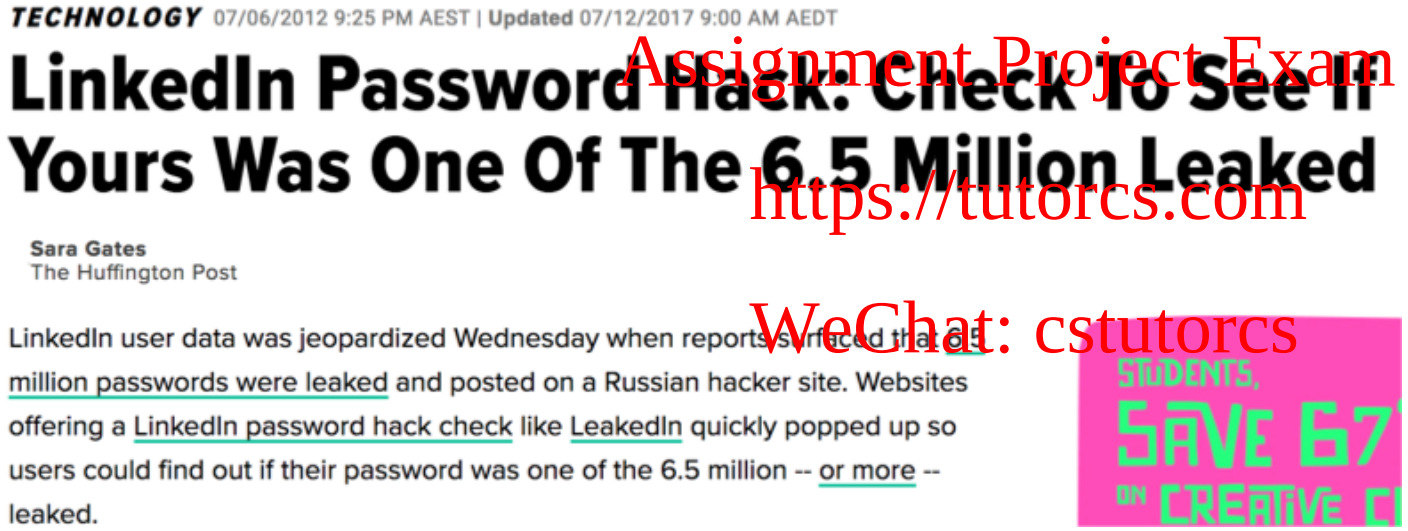


Figure 3.2 UNIX Password Scheme

Real-world Example#

Software & System Security: Entity AUTHentication

- June 2012: 6.5 Million passwords of LinkedIn users posted on Russian hacker site



TECHNOLOGY 07/06/2012 9:25 PM AEST | Updated 07/12/2017 9:00 AM AEDT

LinkedIn Password Hack: Check To See If Yours Was One Of The 6.5 Million Leaked

Sara Gates
The Huffington Post

LinkedIn user data was jeopardized Wednesday when reports surfaced that 6.5 million passwords were leaked and posted on a Russian hacker site. Websites offering a LinkedIn password hack check like LeakedIn quickly popped up so users could find out if their password was one of the 6.5 million -- or more -- leaked.

- Vulnerabilities: Used a general purpose hash function (fast) without salt (low memory precomp.)- vulnerable to precomputation dictionary attack

MONASH University

Password-based AUTH: **Password** Management in UNIX


UNIX Implementation: salted pwd hashing

Software & System Security: Entity AUTHentication

- **Original UNIX scheme (1970's)**
  - 8 character password, 12-bit salt
  - now regarded as very insecure
    - e.g. supercomputer, 50 million tests, 80 min
  - sometimes still used for backward compatibility
- **Modern UNIX (e.g. Mac OS) & Linux distributions:**
  - Typically, unrestricted pwd length, 128-bit salt and SHA-256/512-based one-way func
  - use modern password hashing algorithms such as bcrypt or scrypt

UNIX Implementation: password files


Software & System Security: Entity AUTHentication

- **/etc/passwd**
  - Contains an entry for each user with username & public info, but **not** password
  - Usually readable by all users
    - `other' read permission (see access control slides)
  - 'x' character indicates password hash is stored separately in /etc/shadow file
  - See link on Moodle for typical linux file format

- **/etc/shadow**
  - Contains an entry for each user with username, salt, and salted hash value
  - Restricted to be readable only by root user (to avoid precomp. Dictionary attacks)
  - See link on Moodle for typical linux file format

MONASH University

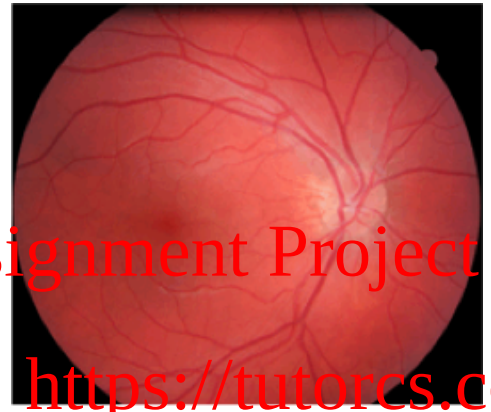
User AUTH Case Study: **Biometrics**

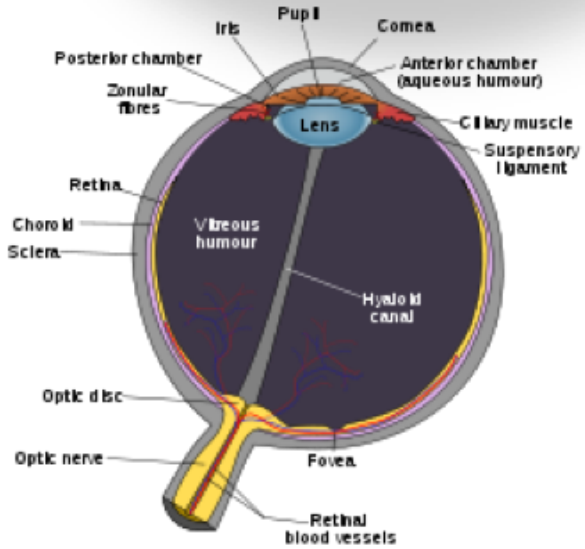
Biometrics: Something You Are (SYA)

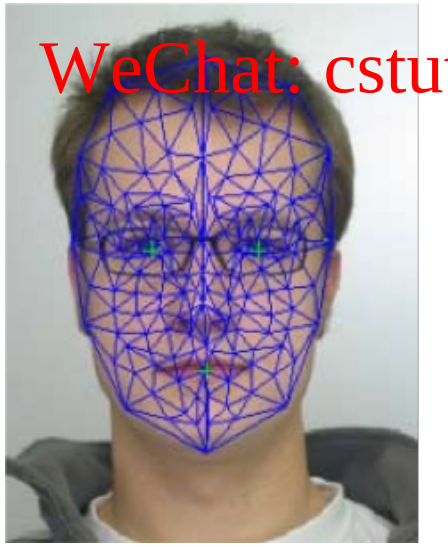
Software & System Security: Entity AUTHentication

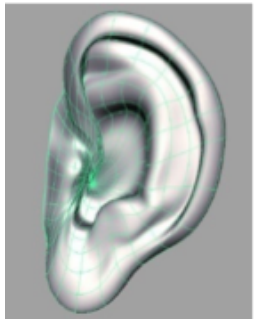
- Physical or behavioural trait of the human body
- presumed to be **unique & invariant** over time
- common biometrics:
  - Fingerprint
  - Iris Scan
  - Retinal/Iris Scan
  - Hand Geometry / Palm
  - Facial recognition
  - Voice, typing, key strokes, signature pressure

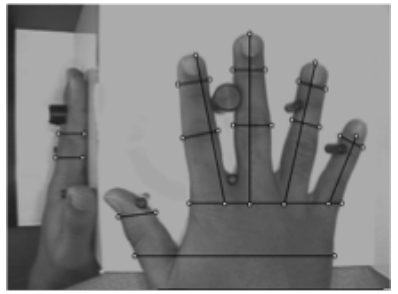
*Examples: Iris recognition, Retinal scan, ...











Static vs Dynamic Biometrics

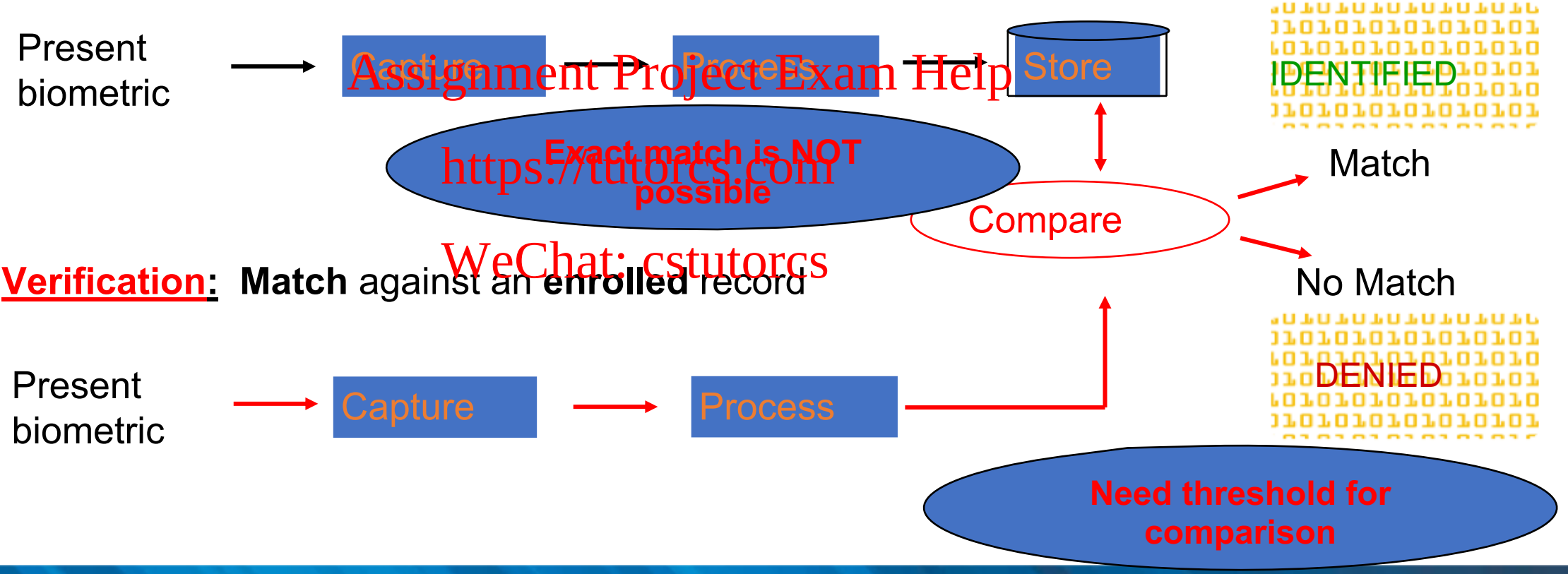
Software & System Security: Entity AUTHentication


- **Static biometric**
  - Does not change from one session to another
  - Relatively **static**
  - Fingerprint, Face, Hand, Iris, Retina
- **Dynamic / Behavioural biometric**
  - Voice, Keystrokes, Gait

Biometrics: How it works

Software & System Security: Entity AUTHentication

Enrollment: **Add** a biometric identifier to a **database**



Verification: **Match** against an **enrolled** record

39

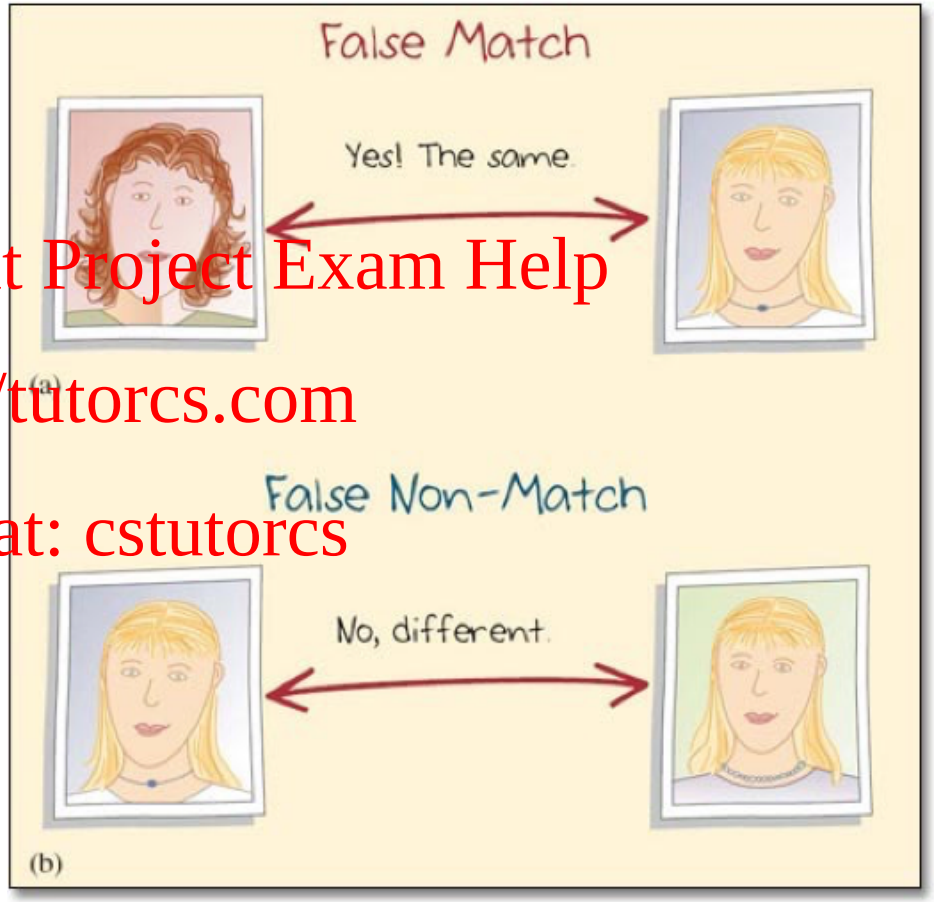
Biometric **Accuracy / Reliability**

Software & System Security: Entity AUTHentication

QUESTION:
Which matching error type is more important for biometric AUTH security and why?

Two types of matching errors:

- false match /accept /positive
  - when biometric data from different people are judged to be from the same person, as in (a)
- false non-match /reject /negative
  - when biometric data from the same person are judged to be from different people, as in (b)



Biometric **Accuracy / Reliability**

Software & System Security: Entity AUTHentication

- False Positive (/Acceptance) Rate (FAR)
  - Measures how often an unauthorized (imposter) user, who should NOT be recognized by the system, is falsely recognized
    - > You are pretending to be me!
- False Negative (/Rejection) Rate (FRR)
  - Measures **how often** an authorized (genuine) user, who should BE recognized by the system, is not recognized
    - > I am not recognised as me!
- Ideally, want both FRR and FAR to be zero,
  - but that is not possible because of the overlapping regions
  - → Need to choose a suitable trade-off!

Biometric **Accuracy / Reliability**

Software & System Security: Entity AUTHentication

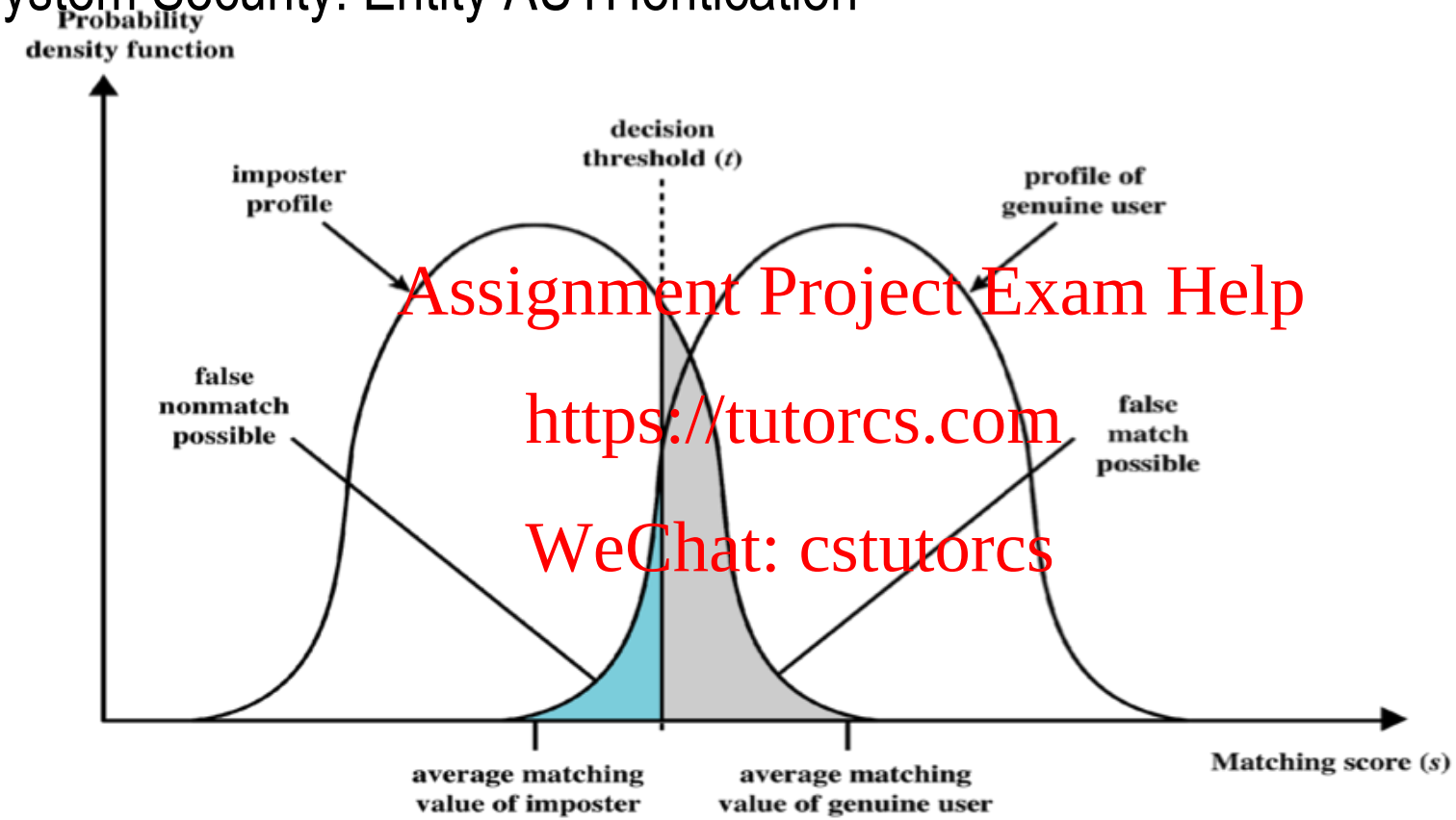


Figure 3.8 Profiles of a Biometric Characteristic of an Imposter and an Authorized Users In this depiction, the comparison between presented feature and a reference feature is reduced to a single numeric value. If the input value (s) is greater than a preassigned threshold (t), a match is declared.


Forgery Attacks on Biometrics: Fingerprints

Software & System Security: Entity AUTHentication

- Assumptions (I) & (II):
  - only you should have it & others would not have it
  - harder to forge, vs SYH

- The case for biometric fingerprints
  - attack on assumptions (I) & (II):
    - fake gummy fingers [Tsu2002]

  - defense against fake finger attacks
    - differentiate real vs fake, via liveness detection
      - liveness: temperature, sweat, pulsation, …
      - used in some industry fingerprint scanners

[Tsu2002] https://doi.org/10.1007/3-540-36178-2_36


Forgery Attacks on Biometrics: Fingerprints

Software & System Security: Entity AUTHentication

- security against fake finger attacks
  - differentiate real vs fake, via liveness detection
    - liveness: temperature, sweat, pulsation, …
    - used in some industry fingerprint scanners

- yet, there are forgery attacks on liveness-detecting fingerprint scanners
  - exploits weakness:
    - liveness detection separate from biometric fingerprint detection, so just need to fool the liveness detector

    - [Bow2012] https://doi.org/10.1007/978-3-642-28368-0_32

Forgery Attacks on Biometrics: Fingerprints

Software & System Security: Entity AUTHentication

- fingerprint lifting & enhancement

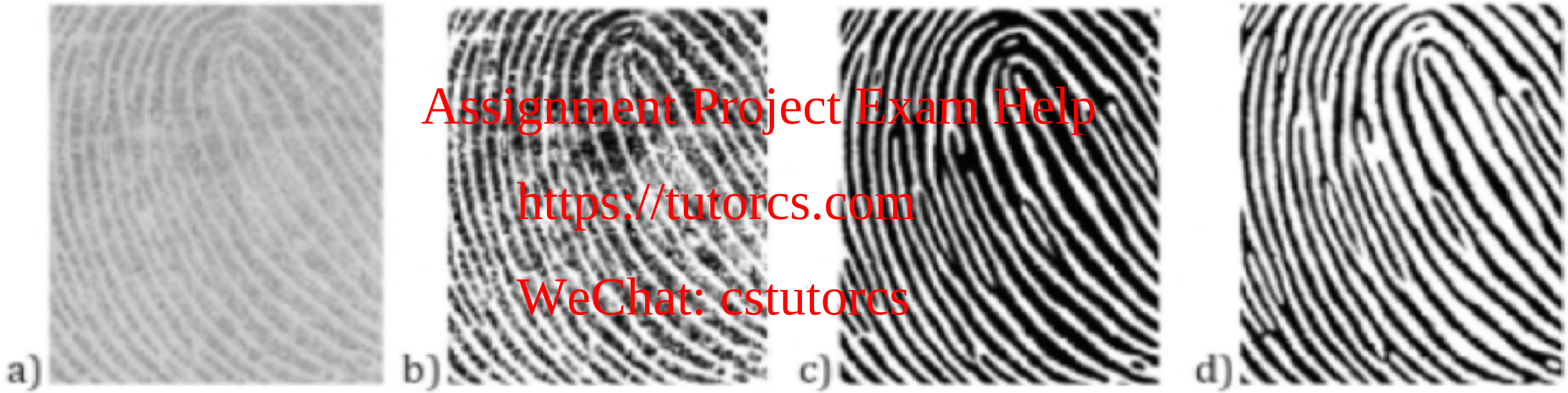
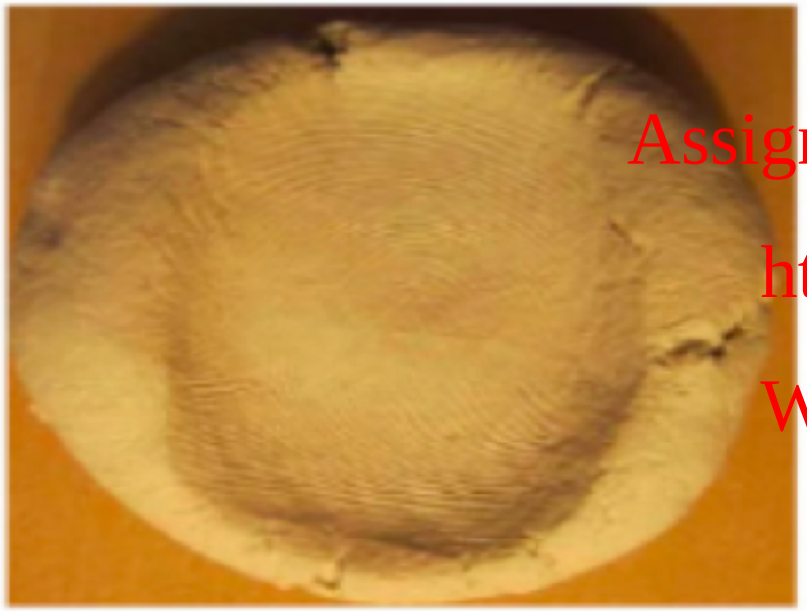


Fig. 1. Different states of the fingerprint image during the cleaning process: (a) scan of dusted fingerprint (b) contrast increased between ridges and valleys (c) ridges drawn over with solid lines (d) negative image

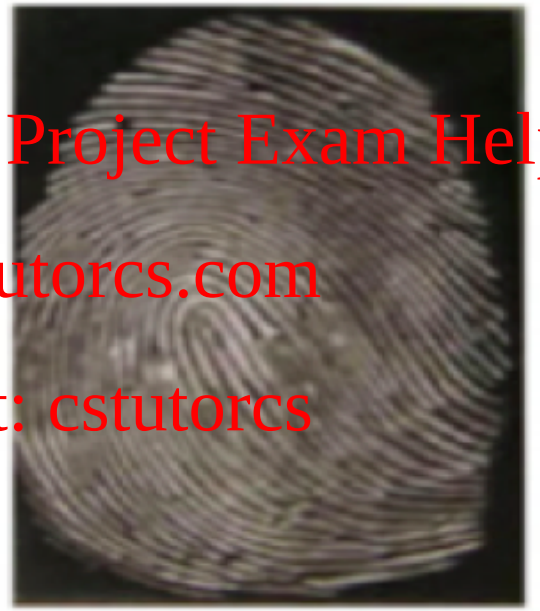
Forgery Attacks on Biometrics: Fingerprints

Software & System Security: Entity AUTHentication

- producing fake fingers with **moulds**



a)



b)

c)


Forgery Attacks on Biometrics: Fingerprints

Software & System Security: Entity AUTHentication

- The case for latent biometric fingerprints
  - latent: fingerprint left on surface / at (crime) scene
- Assumptions (I), (II) & (III):
  - only you should have it, others would not have it, if see it means you were there
- Attack on assumptions (I), (II) & (III):
  - fingerprint transplantation [Ask2018]

- [Ask2018] https://doi.org/10.1049/iet-bmt.2016.0113
- [Youtube] https://www.youtube.com/watch?v=6jDMO8L0raE

Forgery Attacks on Biometrics: Transplantation

Software & System Security: Entity AUTHentication

Forgery Attacks on Biometrics: Transplantation

Software & System Security: Entity AUTHentication

Fingerprint left on surface

Fingerprint dusted

49

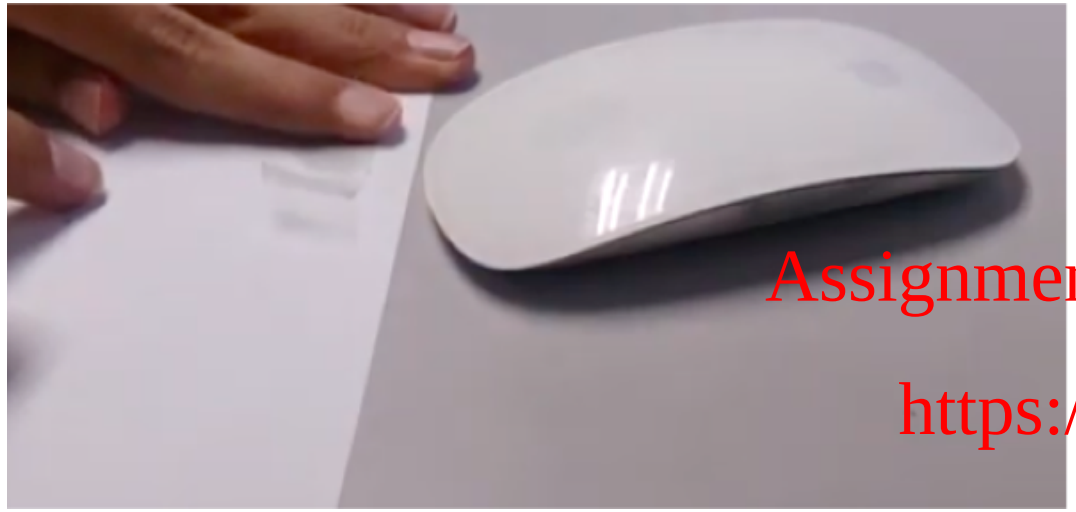
Forgery Attacks on Biometrics: Transplantation

Software & System Security: Entity AUTHentication

Fingerprint lifted with tape

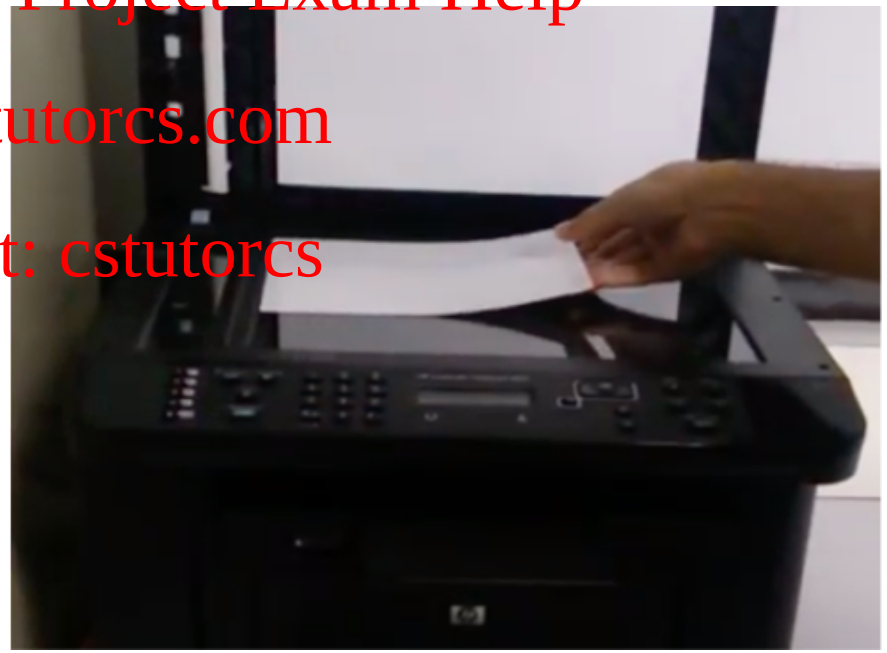
Forgery Attacks on Biometrics: Transplantation

Software & System Security: Entity AUTHentication



Fingerprint pasted on paper

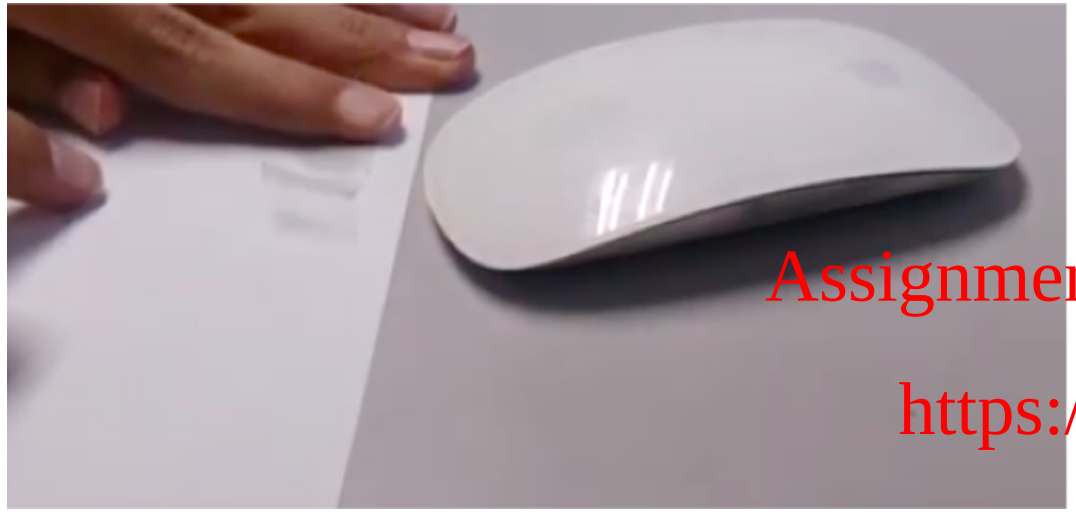
Fingerprint scanned



51

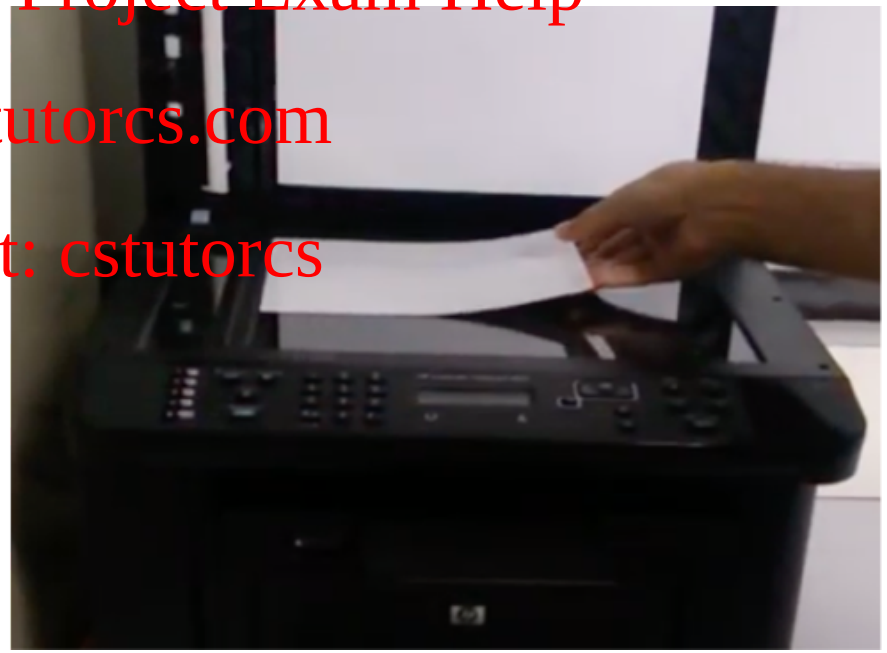
Forgery Attacks on Biometrics: Transplantation

Software & System Security: Entity AUTHentication



Fingerprint pasted on paper

Fingerprint scanned



52

Forgery Attacks on Biometrics: Transplantation

Software & System Security: Entity AUTHentication

Fingerprint planted somewhere else

53


Forgery Attacks on Biometrics: Summary

Software & System Security: Entity AUTHentication

- Assumptions (I), (II) & (III):
  - only you should have it, others would not have it, if see it means you were there
- Attacks on assumptions (I), (II) & (III):
  - fake finger [Tsu2002]
  - fingerprint transplantation [Ask2018]
  - invalidate the assumptions that biometrics unforgeable, irrespective of whether
    - detected at fingerprint scanners, or
    - obtained from (crime) scenes
  - be careful of false sense of security caused by invalid assumptions

Access Control:
Control what a user can do

AUTH & **Access Control**

Software & System Security: Entity AUTHentication


- AUTH
  - link person to ID
  - 1st stage of Access Control
- Access Control: must be **enforced** after AUTH
  - regulate access to (private) resources
  - link ID to what can be accessed & how to access (e.g. read, write)

Access Control Terminology

- **subject - entity that can access objects**
  a process representing user/application
  often have 3 classes: owner, group, world
- **object - access controlled resource**
  e.g. files, directories, records, programs etc
  number/type depend on environment
- **access right - way in which subject accesses an object**
  e.g. read, write, execute, append, delete, create, search

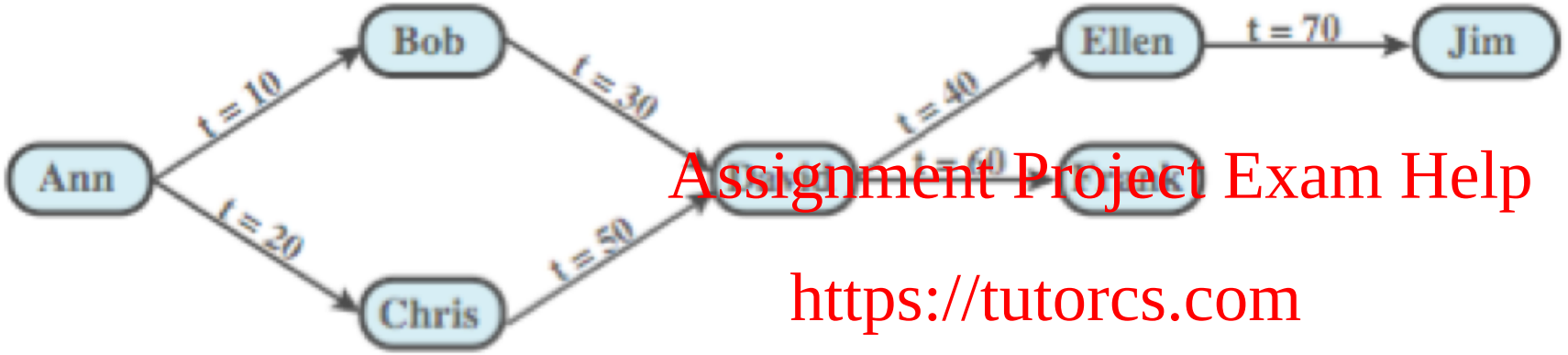
Access Control Delegation

Software & System Security: Entity AUTHentication

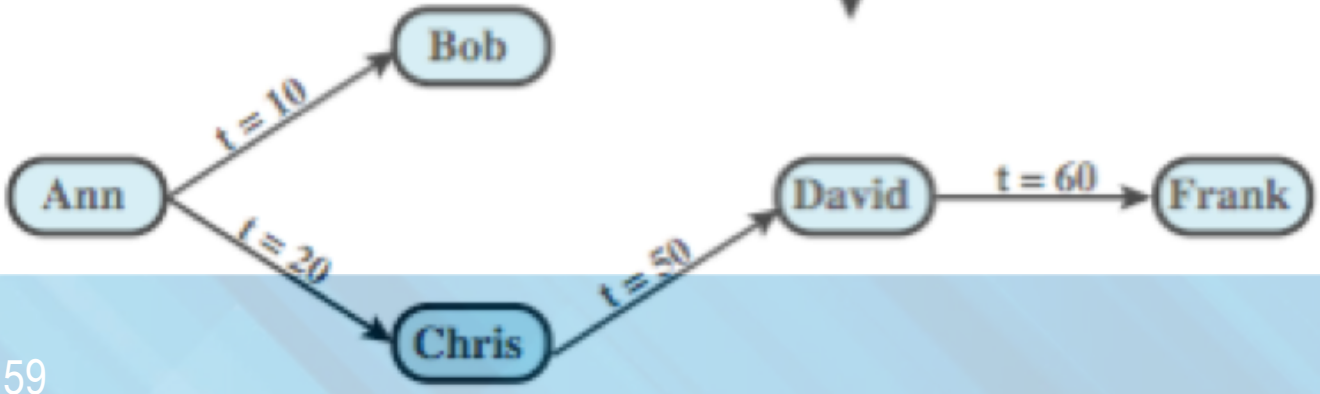

- Access rights can be delegated
- the **propagation of privileges** can be visualized using a diagram, where nodes represent users and directed edges represent granted privileges
- A user, Alice, who has granted privileges to another, Bob, can opt to **revoke** those privileges at a later time

Access Control Delegation

Software & System Security: Entity AUTHentication



- Cascading authorizations



Role-based Access Control (RBAC)


Software & System Security: Entity AUTHentication

- vs grant all access rights so that can run all tasks of that user
  - to run each task requires own set of privileges
- types of users:
  - application **owner**: end user who owns resource as part of an application
  - end **user**: end user who operates on resource objects via a particular application but not own any of them
  - **admin**istrator: user who has administrative responsibility for part or all of the resource
- an RBAC needs to provide the following **capabilities**:
  - **create** and **delete** roles, define **permissions** for a role
  - **assign** and **cancel** assignment of **users to roles**

MONASH University

Access Control Case Study: UNIX Access Control

UNIX file access permissions

- Each file or directory is owned by some user id and a group id.
- File's read (r) / write(w) / execute(x) permissions for owner, group and everyone else ('other') are set in file's **access permissions mode**

UNIX file attributes

- Access permission mode for a file/dir can be displayed with **ls –ls** command:

e.g.

$ ls -ls exercise

2 -rw-r--r-- 1 srini users 1512 Jul 14 13:32 exercise

Explanation:

- permission mode of the file is
  - read/write for owner,
  - read only for the group and
  - read only for others
- user-id of the file owner: *srini*
- group-id of the file: *users*
- size of file is 1512 bytes
- File was last modified on July 14 at 13:32 hours
- The file name is exercise

UNIX file access control security

- Access permission mode for a file/dir can be displayed with ls –ls command:
  - **Important system files must have appropriate file permissions**

e.g:

```
-r--------   1   root sys    /etc/shadow
-rw-r--r--   1   root sys    /etc/profile
drwxr-xr-x   18  root sys    /usr
```

Sensitive password hash info!

UNIX administrator (root) role

- **root** (or superuser) is a privileged administrator user in UNIX.
- root can access all components (files, devices, process, program,…) of the UNIX system.
- → Protection of root password is paramount in UNIX security.
- If a user knows the root password, it can elevate its access control privilieges for a command by running 'sudo [command name]'
- **Q**: what would **sudo cat /etc/shadow** do?

UNIX access control: user/group id

- **When a process executes, it has four id's:**
  - a **real** user id (real uid)
  - an **effective** user id (eff uid)
  - a **real group** id (real gid)
  - an **effective group** id (eff gid)
- When you log on, your shell process has
- real uid = eff uid = user's id
- real gid = eff gid = user's group
- can use **id** command to find out your uid and gid.

- **A process can access a file with permissions:**
  - If the process's effective user id is same as the owner of the file then *User* permissions apply
  - Otherwise, if the process's effective group id is the same as file's group id then *Group* permissions apply
  - Otherwise, *Others* permissions apply

UNIX temporary root privilege: SUID/SGID permission

- **Practical problem:**
- Some files are sensitive (e.g. system password /etc/shadow file)
  - should only be readable/writeable by administrator owner (root user)

- **Q:** What if a (non-root) user Alice wants to change her password?

  → Alice runs the "passwd" program, enters new password

  → **passwd** program should write new password to "shadow" password file

  → **Problem:** If root only allows writing permission on "shadow" to himself, Alice's **passwd** process does not have write permission to "shadow" file!

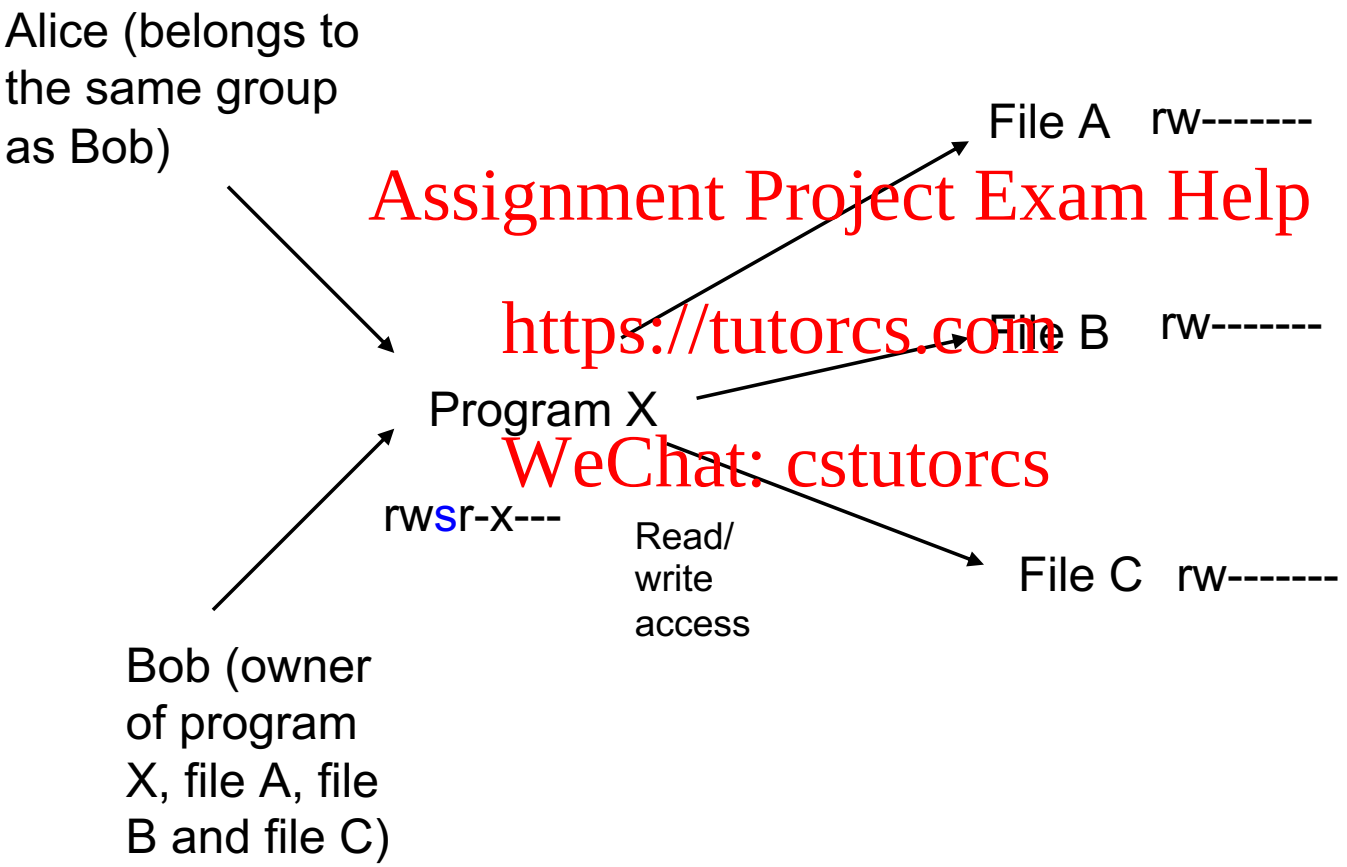
- **Solution in UNIX - SUID/SGID bits:** Allows owner to grant temporary access to files for a specific process (executed program) specified by owner

UNIX temporary root privilege: SUID/SGID permission

- **Access control in UNIX is defined by the owner of the file.**
- **Solution in UNIX - SUID/SGID bits:** Allows owner to grant temporary access to files for a specific process (executed program) specified by owner
- E.g. Owner sets the SUID (or SGID) permission (denoted s) for executable **passwd** file
- Any user Alice running **passwd** receives temporary owner (root) permission
- When **passwd** process runs, effective id = owner's (root) uid, real id = user's (Alice) uid
- → **passwd** process will have access to all the files that the owner (root) can read/write, regardless whether Alice has direct read/write permission to the files.

UNIX temporary root privilege: SUID/SGID permission

Example



QUESTIONS

- Alice can run program X - why?
- Alice cannot directly read/write to Files A/B/C - why?
- But when Alice runs program X, her process running X can access files A/B/C - why?

UNIX SUID/SGID permission: Dangers

- **Q**: What if attacker can modify a program with root SUID?
- **Q**: What if a program with root SUID has a (e.g. buffer overflow) vulnerability?

- Such vulnerability can be used by attacker for privilege escalation – gain unauthorised root permissions! (e.g. access other user's files in the system!)
- **Lessons**:
- Only give SUID permission to trusted and carefully tested programs! Make sure only owner has write permission to such programs!

UNIX Access Control: Common Vulnerabilities

- Incorrect default permissions/configurations:

On most Linux distributions the default permissions for the home folder of each user is 755 (i.e. rwxr-xr-x) – i.e. anyone can read what is in other users' home folders!

In several programming languages, a function that creates a new file or directory (e.g. "mkdir" in PHP) will give by default full access permisions 777 (rwxrwxrwx) to all users.

Hackers look for such misconfigured resources!
Lesson: Do not rely on default access permissions – set them appropriately!

Further Reading

- Chapters 3 & 4 of the textbook: *Computer Security: Principles and Practice" by William Stallings & Lawrie Brown,* Prentice Hall, 2015
- Chapter 8 of the book: "*Practical Unix & Internet Security"* by Simon Garfunkel and Gene Spafford, Edition 2, 2003
- Common Linux access control mis-configurations:
  - https://resources.securitycompass.com/blog/5-common-linux-misconfigurations-2
- Password hashing security:
  - https://crackstation.net/hashing-security.htm
- Password key derivation hash function scrypt:
  - http://www.tarsnap.com/scrypt/scrypt.pdf