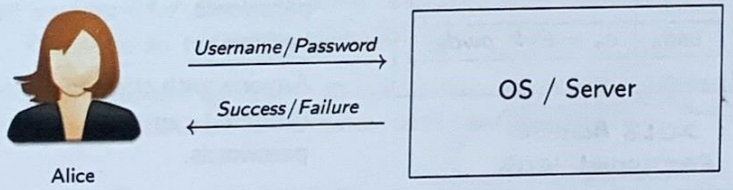
Password authentication

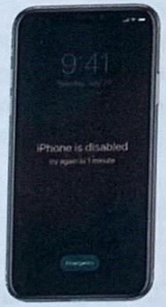
- The question: "who is allowed to access the resources in a computer system?"
- How does the operating system securely identify its users?
- Authentication: determination of the identity of a user
- Standard authentication mechanism: **username** and **password**



Passwords need to be hard to guess yet easy to remember.

Defending against online guessing attacks

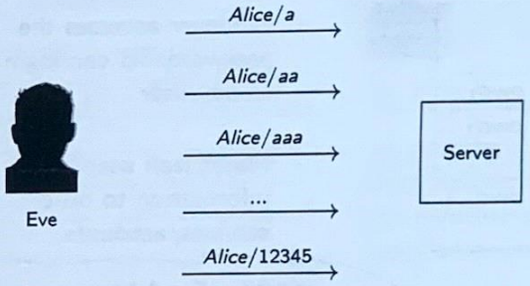
1) **Rate limit** - impose a limit on the number of failed password attempts before locking the system for a set amount of time ↳ Slow down the attacker



2) Include **captchas** - include a captcha puzzle to be solved along the submission of the username and password in order to prevent automated password guessing

assuming computers are not good at solving captchas

Online guessing attacks



Offline guessing attacks

- Most common password-related attacks target the server

access the pswd dbase / shadow file

Our goal

Defend from attacks that leak the password database

Attempt #1: store passwords **unencrypted**

NOT GOOD!!!

Password DB

usn_1	pwd_1
usn_1	pwd_2
...	...
usn_n	pwd_n

- Whoever accesses the password DB can login as any user
- Might leak user login information to other services/accounts

2006 Reddit password leak

Many users reuse password across multiple applications!

Attempt #2: **encrypt** passwords

Password DB

k

usn_1	$c_1 = E(k, pwd_1)$
usn_2	$c_2 = E(k, pwd_2)$
...	...
usn_n	$c_n = E(k, pwd_n)$

2013 Adobe Password leak

+ Stolen encrypted passwords cannot be decrypted,
+ Only admins have the key. If a user forgets their password, admins can just look it up for him,

- If attacker managed to steal passwords, why assume the key cannot be stolen?
- Anyone with the key (admins) can view passwords.

We pushed the problem into the key.

STILL NOT GOOD!!

Attempt #3: **hash** passwords

? Stolen hashed passwords cannot easily be cracked (?!)

Password DB

usn_1	$d_1 = H(pwd_1)$
usn_2	$d_2 = H(pwd_2)$
...	...
usn_n	$d_n = H(pwd_n)$

2012 LinkedIn Password Leak

- Once a hash is cracked, the password is know for all accounts using the same password → Users w/ the same pswd. are associated w/ the same hash
- Humans tend to pick weak/guessable passwords
  - Frequency analysis
  - Dictionary attack

PASSWORD CRACKING

1. Brute force attack

- Try all passwords in a given space
  - κ: number of possible characters
  - ℓ: password length
  - $\rightsquigarrow$ κ^ℓ possible passwords

Tips for safe (strong) passwords

Hackers are very good at finding out passwords. They don't simply try to guess them, they get very fast computer programs to try out millions, very quickly. Hackers also know the kind of "tricks" that people use to try to strengthen their passwords.

We advise you memorise a few strong passwords for the systems you use regularly. For services you use less often, find a way to manage those passwords that works for you so that you can look them up, or work them out when you need them.

- University systems require a password length of seven. We recommend you choose more. See "Long passwords" below.
- Use a mix of upper- and lower-case letters, numbers and punctuation marks
- A strong password looks like a random sequence of symbols - use some non-alphabetic characters such as @#$!%+-/:?_
- Use non-dictionary words - like XKCD or one of the other approaches, described below

UoE password guidelines

$\rightsquigarrow$ Assuming a standard 94 characters keyboard, there are $94^7 = 6.4847759e^{+13}$ possible passwords.

Does an attacker need to try all 94^7 passwords?

↳ NO! The effective/most common password space $\ll$ all possible pswd.

(2) Dictionary attack

↗ is very efficient bc. users tend to pick weak passwords!

- Try the top N most common passwords,
- Try words in English dictionary,
- Try names, places, notable dates,
- Try Combinations of the above,
- Try the above replacing some characters with digits and symbols e.g. : iloveyou, il0vey0u, i10v3y0u, ...
- UoE: password guidelines `https://www.ed.ac.uk/infosec/how-to-protect/lock-your-devices/passwords`

to prevent this...

Attempt #4: **salt and hash** passwords

Introduce randomness in the way pswd is stored

Password DB

random salt

usn_1	s_1	$d_1 = H(s_1 \| pwd_1)$
usn_2	s_2	$d_2 = H(s_2 \| pwd_2)$
...	...	
usn_n	s_n	$d_n = H(s_n \| pwd_n)$

Salt is stored

+ Since every user has different salt, identical passwords will not have identical hashes ✱ — considerably slows down attackers!!
+ No frequency analysis
+ No precomputation: when salting, one cannot use preexisting tables to crack passwords easily — w/o salt, the attacker can precompute even before any password leak

- store **salted hashes** of passwords
- use a **slow hash function** - eg. $H(pwd) = h^{1000}(pwd)$

iterate hash fn 1000 times, hence attack will be 1000 times slower!

but there's a tradeoff to authenticate the user

Two factor authentication (2FA)

- **Malware attack** - users will often have malware installed on their machine - this malware might contain a key-logger that records keyboard stroke and intercept passwords when typed
- Key-logger mitigation - use **two factor authentication** (2FA)

SOLN

A device user is granted access only after successfully presenting 2 pieces of evidence to an authentication mechanism
- KNOWLEDGE (smthg only user knows)
- POSSESSION (smthg only user has)
- INHERENCE (smthg only user is)

Password manager

- **Strong passwords are not easy to remember** - users are expected to memorise tens of different hard to guess passwords and humans are not good at this
- Weak passwords mitigation - use a **password manager** - pick and memorise a single strong password to the password managers which takes care of storing and managing all the other passwords