

Lecture 14: Passwords

CS 138

Spring 2026

<p>UNCOMMON (NON-GIBBERISH) BASE WORD</p> <p>ORDER UNKNOWN</p> <p>Tr0ub4dor&3</p> <p>CAPS? COMMON SUBSTITUTIONS</p> <p>NUMERAL PUNCTUATION</p> <p>(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)</p>	<p>~28 BITS OF ENTROPY</p> <p>$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$</p> <p>(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE: YES, CRACKING A STORED HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)</p> <p>DIFFICULTY TO GUESS: EASY</p>	<p>WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?</p> <p>AND THERE WAS SOME SYMBOL...</p> <p>DIFFICULTY TO REMEMBER: HARD</p>
<p>correct horse battery staple</p> <p>FOUR RANDOM COMMON WORDS</p>	<p>~44 BITS OF ENTROPY</p> <p>$2^{44} = 530 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$</p> <p>DIFFICULTY TO GUESS: HARD</p>	<p>THAT'S A BATTERY STAPLE.</p> <p>CORRECT!</p> <p>DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT</p>

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Where we were...

- **Something you are**
fingerprint, retinal scan, hand silhouette, a pulse
- **Something you know**
password, passphrase, PIN, answers to security questions
- **Something you have**
physical key, ticket, {ATM, prox, credit} card, token

Password lifecycle

1. **Create:** user chooses password
2. **Store:** system stores password with user identifier
3. **Use:** user supplies password to authenticate
4. **Change/recover/reset:** user wants or needs to change password

1. PASSWORD CREATION

Who creates?

- **User**
- **System**
- **Administrator**

Exercise 1: Choosing Passwords

Guess the top five most common US passwords in 2025

Measuring Password Strength

Entropy

- Difficulty to brute force—
"strength" or "security level"
 - Recall: if 2^X guesses required, strength is X
- If L characters long with N possible characters:
 - Strength = $L \log_2 N$

Assuming every password is equally likely!

NIST Entropy Estimates (2006)

- 1st character: 4 bits
- next 7 characters: 2 bits per char
- characters 9..20: 1.5 bits per char
- characters 21+: 1 bit per char
- user forced to use lower & upper case and non-alphabets: flat bonus of 6 bits
- prohibition of passwords found in a 50k word dictionary: 0 to 6 bits, depending on password length

Turns out to be bad predictor of how quickly attacker can guess!

Measuring Password Strength

Password Cracking

- Evaluate recipes based on
 - percentage of passwords cracked
 - number of guesses required to crack

This is state of the art today

Idea: Password Recipes

- **Problem:** guide users into choosing strong passwords
- **Solution:** **password recipes** are rules for composing passwords
 - e.g., must have at least one number and one punctuation symbol and one upper case letter

CREATE YOUR USERNAME *

CREATE YOUR PASSWORD *

 Show

Your password must

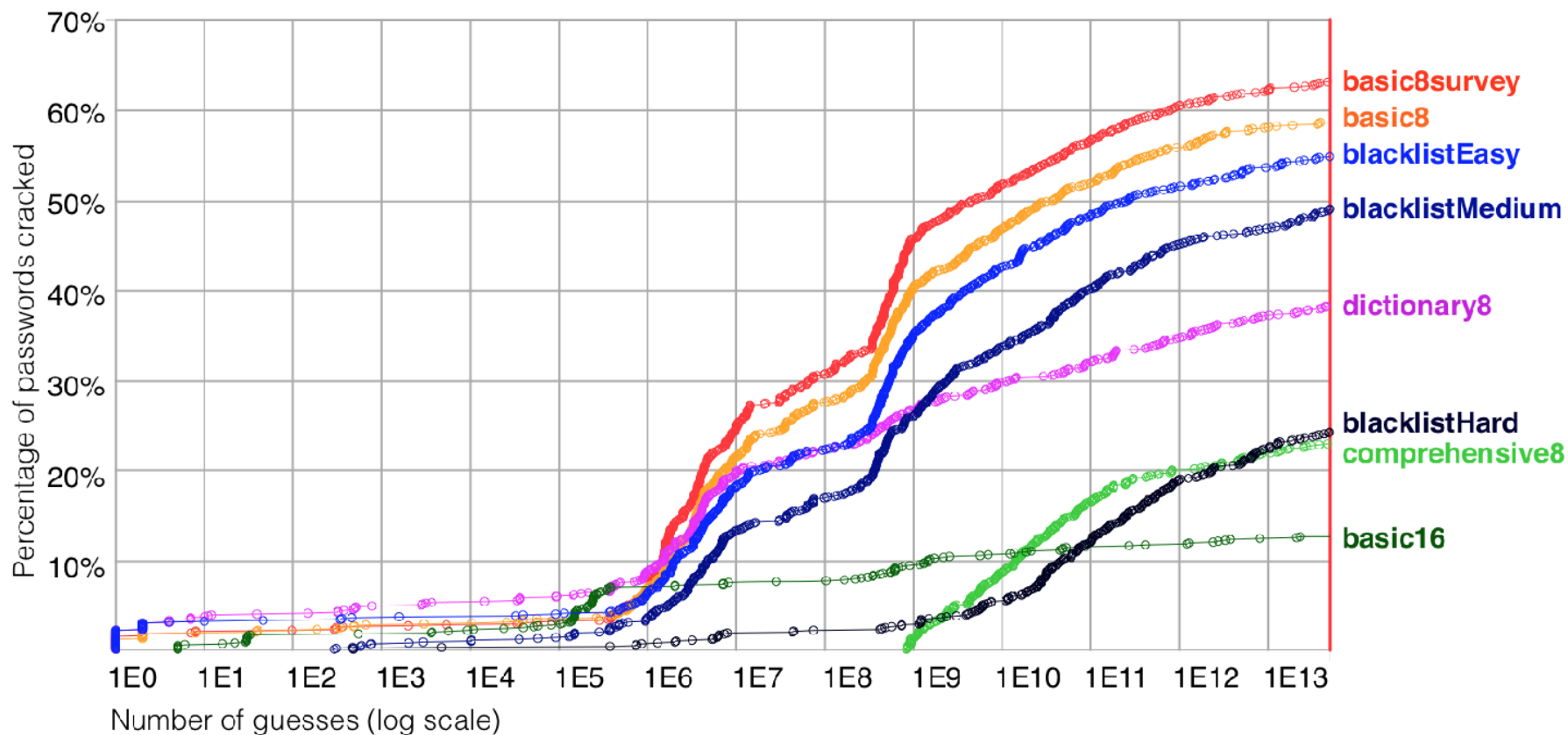
- Be at least 9 characters
- Include an uppercase letter
- Include a lowercase letter
- Include a number
- Not start or end with a space

CREATE YOUR CALL-IN PIN *

Evaluating Password Recipes

- Evaluate recipes based on
 - percentage of passwords cracked
 - number of guesses required to crack
- Example recipes:
 1. ≥ 8 characters
 2. ≥ 8 characters, no blacklisted words ...with various blacklists
 3. ≥ 8 characters, no blacklisted words, one uppercase, lowercase, symbol, and digit ("comprehensive", c8)
 4. ≥ 16 characters ("passphrase", b16)
- Results...

Recipe comparison



User Perceptions of Password Security

iloveyou88

ieatkale88



Study participants' perceptions

iloveyou88

=

ieatkale88

Reality

iloveyou88

ieatkale88

**4,000,000,000 ×
more secure!**



Which is more secure?

brooklyn16

brooklynqy



Study participants' perceptions

brooklyn16

brooklynqy

Reality

brooklyn16

brooklynqy



**300,000 ×
more secure!**

Which is more secure?

sponge01bob

spongebob01



Study participants' perceptions

sponge01bob

spongebob01

Reality

sponge01bob

spongebob01

**900,000 ×
more secure!**

Which is more secure?

1qaz2wsx3edc

thefirstkiss



Study participants' perceptions

1qaz2wsx3edc

thefirstkiss

Reality

1qaz2wsx3edc

thefirstkiss

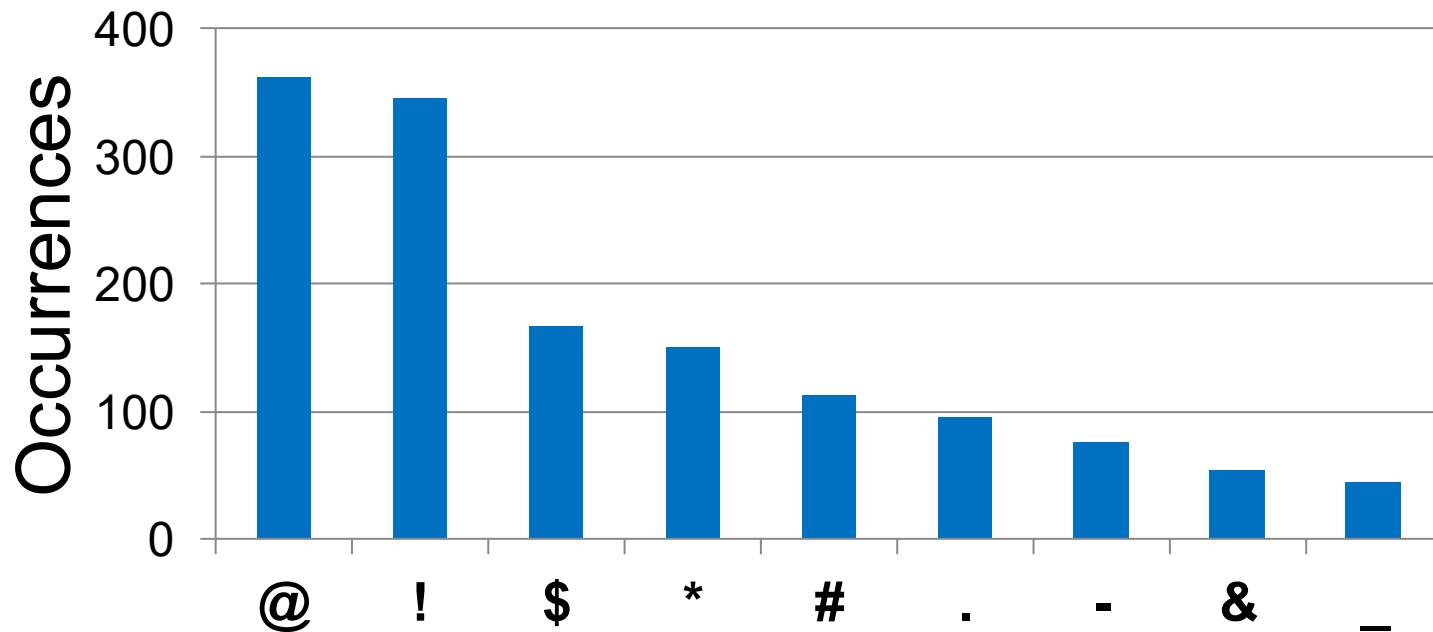
Both are pretty bad!

300×
more secure!



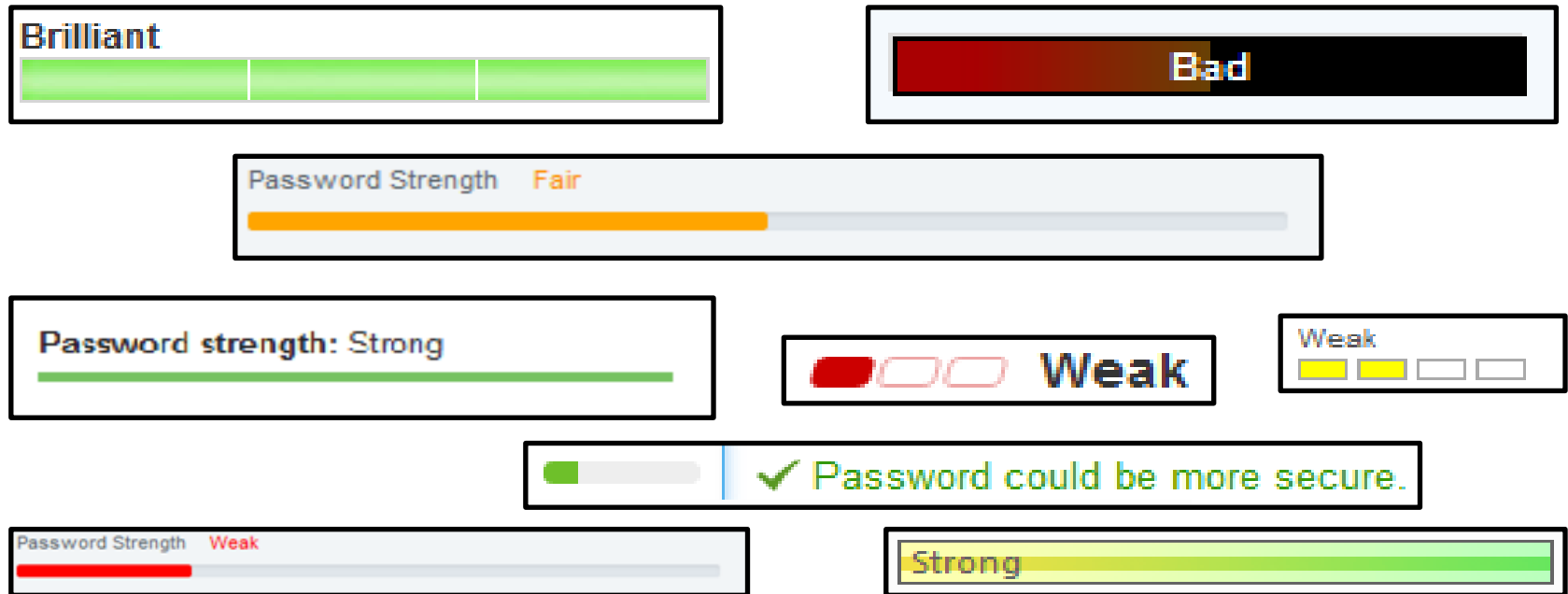
Problem: Humans are predictable

- We all put numbers/special characters in the same place:
 - capital letters usually at the beginning
 - numbers/special characters usually at the end
- We all choose the same character:



Idea 2: Provide Guidance

- Use a visual password meter to give users feedback about how strong their passwords are



- Effectiveness depends on both aesthetics and strictness, but can be very helpful

Passwords

NIST (2017, updated 2020) recommends:

- minimum of 8 characters
- up to 64 characters should be accepted
- all printable ASCII characters and Unicode should be accepted
- blacklist compromised values, dictionary words, repetitive characters, and context-specific words
- no other security requirements

Should provide guidance on picking a good password (e.g., password meter)

2. PASSWORD STORAGE

Password Storage

- Passwords typically stored in a file or database indexed by username
- **Strawman idea:** store passwords in plaintext
 - requires perfect authorization mechanisms
 - requires trusted system administrators
 - ...

Threat Model: Offline Attack



- Adversary can read files from disk

BRANDS WITH 100M+ LEAKED RECORDS

BRAND NAME	RECORDS LEAKED
Tencent	1.5B
Weibo	504M
MySpace	360M
Twitter	281M
Wattpad	271M
NetEase	261M
Deezer	258M
LinkedIn	251M
AdultFriendFinder	220M
Zynga	217M
Luxottica	206M
Evite	179M
Zing	164M
Adobe	153M
MyFitnessPal	151M
Canva	143M
JD.com	142M
Badoo	127M
VK	101M
Youku	100M

cybernews

- Adversary can read process memory

Note: users make this worse by reusing passwords across systems.

Password Storage

- **Want:** a function f such that...
 1. easy to compute and store $f(p)$ for a password p
 2. hard given disclosed $f(p)$ for attacker to recover p
 3. hard to trick system by finding password q s.t. $q \neq p$ yet $f(p) = f(q)$
- Encryption would work, but then the key has to live somewhere
- Cryptographic hash functions work!
 - one-way property gives (1) and (2)
 - collision resistance gives (3)

Hashed passwords

- Each user has:
 - username uid
 - password p
- System stores: uid, $H(p)$

Exercise: Hashed Passwords

- Consider an alternative authentication protocol where user sends uid, $H(p)$ and the service compares $H(p)$ to the stored hash. Would this be more or less secure than sending the plaintext password? Why?

Hashed passwords are still vulnerable















































Assume: attacker does learn password file (*offline guessing attack*)

- Hard to invert: i.e., given $H(p)$ to compute p
- But what if attacker didn't care about inverting hash on arbitrary inputs?
 - i.e., only have to succeed on a small set of p 's: p_1, p_2, \dots, p_n
- Then attacker could build a **dictionary**...

Dictionary attacks

Dictionary:

- $p_1, H(p_1)$
 - $p_2, H(p_2)$
 - ...
 - $p_n, H(p_n)$
- Dictionary attack: lookup $H(p)$ in dictionary to find p
 - And it works because most passwords chosen by humans are from a relatively small set

 711,477,622 Onliner Spambot accounts 	 855,249 Manga Traders accounts
 593,427,119 Exploit.In accounts 	 830,155 Pokémon Negro accounts
 457,962,538 Anti Public Combo List accounts 	 819,478 Warframe accounts
 393,430,309 River City Media Spam List accounts 	 800,157 Onverse accounts
 359,420,698 MySpace accounts	 790,724 Brazzers accounts 
 234,842,089 NetEase accounts 	 777,387 Black Hat World accounts
 164,611,595 LinkedIn accounts	 776,125 Abandonia accounts
 152,445,165 Adobe accounts	 745,355 Android Forums accounts
 112,005,531 Badoo accounts  	 738,556 WildStar accounts
 105,059,554 B2B USA Businesses accounts 	 735,405 MALL.cz accounts
 93,338,602 VK accounts	 709,926 PoliceOne accounts
 91,890,110 Youku accounts	 707,432 Programming Forums accounts
 91,436,280 Rambler accounts	 699,793 mSpy accounts
 85,176,234 Dailymotion accounts	 660,305 CrackingForum accounts
 80,115,532 2,844 Separate Data Breaches accounts 	 657,001 PokéBip accounts
 68,648,009 Dropbox accounts	 648,231 Domino's accounts
 65,469,298 tumblr accounts	 637,340 DaFont accounts
	 620,677 Final Fantasy Shrine accounts
	 616,882 Comcast accounts

Typical passwords

- 7-9 character **root** plus a 1-3 character **appendage**
 - Root typically pronounceable, though not necessarily a real word
 - Appendage is a suffix (90%) or prefix (10%)
- Dictionary of 1000 roots plus 100 suffixes (= 100k passwords) cracks about 24% of all passwords
- More sophisticated dictionaries crack about 60% of passwords within 2-4 weeks
- Given biographical data (zip code, names, etc.) and other passwords of a user...
 - success rate goes up a little
 - time goes down to days or hours

Salted hashed passwords

- **Vulnerability:** one dictionary suffices to attack every user
- **Vulnerability:** passwords chosen from small space
- **Countermeasure:** include a **unique system-chosen value** as part of each user's password

Salted hashed passwords

- Each user has:
 - username uid
 - unique salt s
 - password p
 - System stores: uid, s, H(s, p)
-
1. `Hu->L: uid, p`
 2. `L and S: establish secure channel`
 3. `L->S: uid, p`
 4. `S: let h = stored hashed password for uid;
let s = stored salt for uid;
if h = H(s, p)
then uid is authenticated`

3. PASSWORD USAGE

Threat Model: Online Attack

- Adversary can interact with the server as a user



Bank of America  Higher Standards

Online Banking

Sign In

Enter Online ID:

(5 - 25 numbers and/or letters)

Save this online ID [\(How does this work?\)](#)

Enter Passcode:

(4 - 12 numbers and/or letters)

[Sign In](#)

[Reset passcode](#)

[Forgot or need help with your ID?](#)

Not using Online Banking?

[Enroll now for Online Banking](#) >>

[Learn more about Online Banking](#) >>

[Service Agreement](#) >>

[Pay By Phone user's guide](#) >>

[Go to Online Banking for a state other than California](#)



**Stop writing checks
and you could save \$53**

[Learn more >>](#)

Secure Area

[Home](#) • [Locations](#) • [Contact Us](#) • [Help](#) • [Sign in](#) • [Site Map](#)
[Personal Finance](#) • [Small Business](#) • [Corporate & Institutional](#)
[About the Bank](#) • [In the Community](#) • [Finance Tools & Planning](#) • [Privacy & Security](#)

Bank of America, N.A. Member FDIC. Equal Housing Lender 
© 2010 Bank of America Corporation. All rights reserved.

Official Sponsor 2000-2004
U.S. Olympic Teams 

When authentication fails

- **Guiding principle:** the system might be under attack, so don't make the attacker's job any easier
- Don't leak valid usernames:
 - Prompt for username and password in parallel
 - Don't reveal which was bad
- Record failed attempts and review
 - Perhaps in automated way by administrators
 - Perhaps manually by user at next successful login
- Lock account after too many attempts
- Rate limit login

Rate limiting

- **Vulnerability:** hashes are easy to compute
- **Countermeasure:** hash functions that are slow to compute
 - Slow hash wouldn't bother user: delay in logging hardly noticeable
 - But would bother attacker constructing dictionary: delay multiplied by number of entries
 - Ideally, enough to make constructing a large dictionary prohibitively expensive
- Examples: bcrypt, scrypt, Argon2,...

Slowing down fast hashes

- Given a fast hash function...
- Slow it down by iterating it many times:

```
z1 = H(p) ;
```

```
z2 = H(p, z1) ;
```

```
...
```

```
z1000 = H(p, z999) ;
```

```
output z1 XOR z2 XOR ... XOR z1000
```

- Number of iterations is a parameter to control slowdown
 - originally thousands
 - current thinking is 10s of thousands
- Aka [key stretching](#)

Salt and pepper

- Each user has:
 - username uid
 - unique salt s_1
 - unique pepper s_2
 - password p
- System stores: uid, s_1 , $H(s_1, s_2, p)$

4. PASSWORD CHANGE

Password change

Motivated by...

- **User** forgets password (maybe just *recover* password)
- **System** forces password expiration
 - Naively seems wise
 - Research suggests otherwise
- **Attacker** learns password:
 - **Social engineering**: deceitful techniques to manipulate a person into disclosing information
 - **Online guessing**: attacker uses authentication interface to guess passwords
 - **Offline guessing**: attacker acquires password database for system and attempts to crack it

Change mechanisms

- Tend to be **more vulnerable** than the rest of the authentication system
 - Not designed or tested as well
 - Have to solve the authentication problem without the benefit of a password
- Two common mechanisms:
 - Security questions
 - Emailed reset

Security questions

- Something you know: attributes of identity established at enrollment
- **Pro:** you are unlikely to forget answers
- **Assumes:** attacker is unlikely to be able to answer questions
- **Con:** might not resist targeted attacks
- **Con:** linking is a problem; same answers re-used in many systems

Emailed reset

- Might be your old password (bad!), a new temporary password, or just a reset link
 - **one-time password:** valid for single use only, maybe limited duration
- **Assumes:** attacker is unlikely to have compromised your email account
- **Assumes:** email service correctly authenticates you

Password lifecycle

1. **Create:** user chooses password
2. **Store:** system stores password with user identifier
3. **Use:** user supplies password to authenticate
4. **Change/recover/reset:** user wants or needs to change password

Beyond passwords?

- Passwords are tolerated or hated by users
- Passwords are plagued by security problems
- **Can we do better?**
- Criteria:
 - Security
 - Usability
 - Deployability

Schemes to replace passwords

- Graphical
- Cognitive
- Visual cryptography

Schemes to improve passwords

- Password managers
- Single Sign-On
- Passkeys
- Two-factor authentication

Exercise: Authentication Examples

- Choose an example website (e.g., email provider, social network, or a payments app) and investigate how they handle authentication.
- What are their restrictions on password selection? Do they provide feedback on password strength? Do they support SSO? How do they handle recovery? Do they rely exclusively on passwords?

Something you know

