CY 2550 Foundations of Cybersecurity

Passwords and Authentication

Alina Oprea
Associate Professor, Khoury College
Northeastern University

Outline

- Password authentication, storage
- Biometrics, second factors
- Distributed authentication

Announcements:

- Crypto homework due on Feb. 14
- Holiday: Monday, Feb. 17
- Midterm exam: February 20 in class
- Ethics session
    - February 24 and 27

Midterm exam

- Topics
  - Threat modeling, different types of attackers (eavesdroppers, MitM, passive, active)
  - Secure encryption (perfect security, OTP, CPA secure encryption, modes of operation, randomization)
  - Public-key encryption (RSA)
  - Hash functions (collision-resistance)
  - Key exchange (Diffie-Hellman or using public-key encryption)
  - Integrity checks (MACs and signatures)
  - Certificate authorities and PKI
  - TLS (high-level how it works, handshake and record protocols)
  - Password authentication (storage, attacks on passwords, multiple factors, good/bad password strategies)
- What you can bring: calculator, one-page cheat sheet (letter size)

TLS Threat Modeling

Attacker	Action	Mitigation	Assumption
Eavsdropper	Learns confidential information	Secure encryption	Encryption is CPA secure
MitM	Impersonate server	Certificates and PKI	CAs are trusted
MitM	Modify messages	Integrity checks (MACs and signatures)	MACs and signatures are secure
MitM	Replay old valid messages	Sequence numbers used when computing MACs	Client and server maintain sequence numbers MACs are secure

Types of Secrets

- Actors provide their secret to log-in to a system
- Three classes of secrets:
  1. Something you know
     - Example: a password or PIN
  2. Something you have
     - Examples: a smart card, smart phone, or hardware token
  3. Something you are
     - Examples: fingerprint, voice scan, iris scan
- Combination of methods (multi-factor authentication)

Password Storage So Far

1. Never store passwords in plain text
2. Password recovery from hashed passwords can be done with dictionary attacks for common passwords
   - Select dictionary words and common passwords (from leaked datasets)
   - Add common modifications (digits at the end, character-to-digit substitution)
3. Can use pre-computed rainbow tables for cracking uncommon passwords
   - Compute hash chains and store beginning and end
   - Once a match is found on last column, computation is done from password
   - Storage – computation tradeoff
   - Tables available for passwords up to 14 characters

Hardening Password Hashes

- Key problem: cryptographic hashes are deterministic
  - H(‘p4ssw0rd’) is the same every time it’s computed
  - This enables attackers to build and store lists of hashes
- Solution: make each password hash unique
  - Add a random salt to each password before hashing
  - H(salt + password) = password hash
  - Each user has a unique random salt
  - Even when passwords are the same, the salt makes the hash different
  - Salts can be stored in plain text

Example Salted Hashes

hashed_password.txt	
charlie	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

hashed_and_salted_password.txt		
charlie	a8	af19c842f0c781ad726de7aba439b033
sandi	0X	67710c2c2797441efb8501f063d42fb6
alice	hz	9d03e1f28d39ab373c59c7bb338d0095
bob	K@	479a6d9e59707af4bb2c618fed89c245

Breaking Hashed Passwords

- **Stored passwords should always be salted**
  - Forces the attacker to brute-force each password individually
- Problem: it is now possible to compute hashes very quickly
  - GPU computing: hundreds of small CPU cores
  - nVidia GeForce GTX Titan Z: 5,760 cores
  - GPUs can be rented from the cloud very cheaply
    - $0.9 per hour (2018 prices)
- Example of hashing speed
  - A modern x86 server can hash all possible 6 character long passwords in 3.5 hours
  - A modern GPU can do the same thing in 16 minute

Hardening Salted Passwords

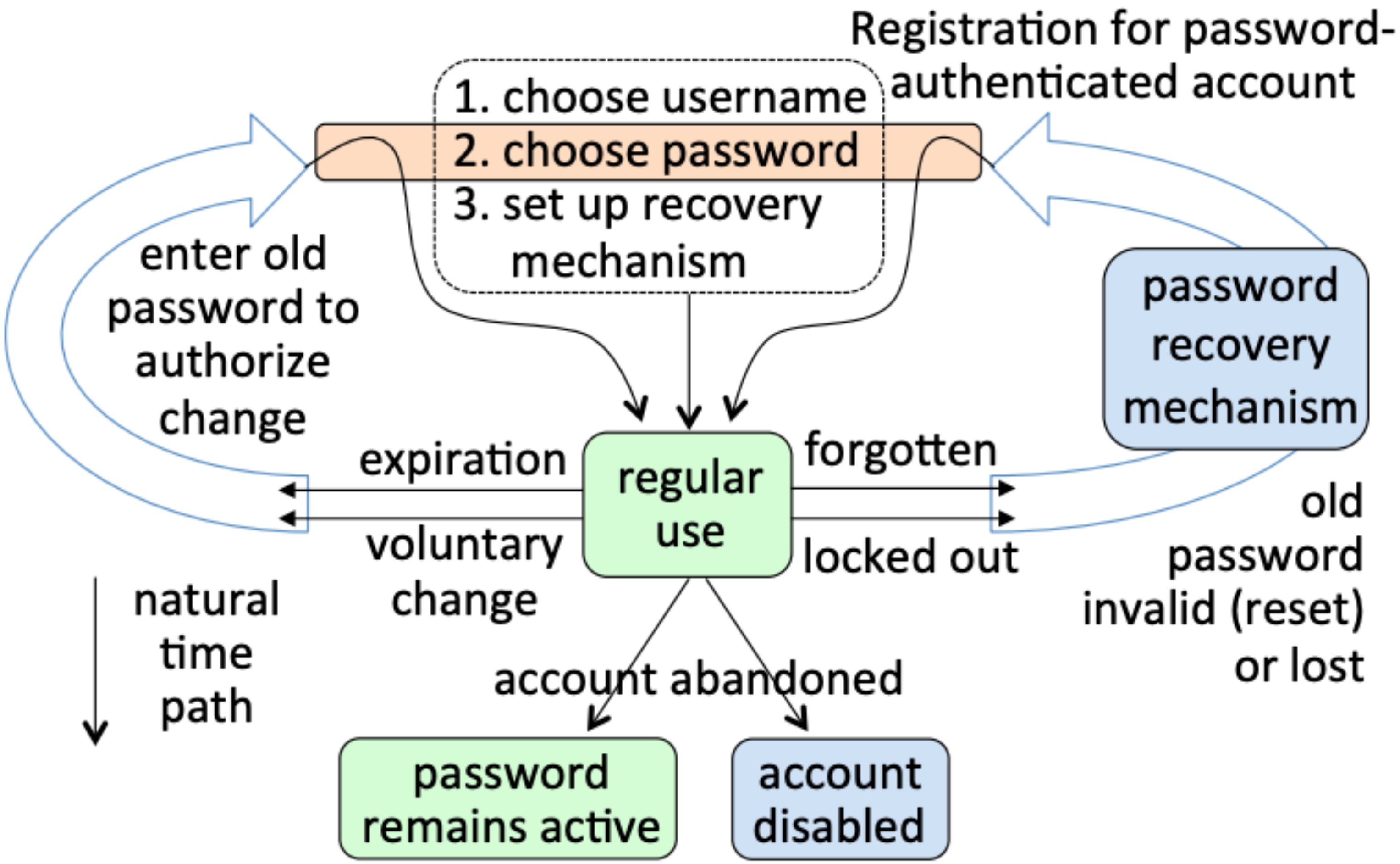
- Problem: typical hashing algorithms are too fast
  - Enables GPUs to brute-force passwords
- Old solution: hash the password multiple times
  - Known as key stretching (compute multiple iterations of hashing)
- New solution: use hash functions that are designed to be **slow**
  - Examples: bcrypt, PBKDF2, scrypt
  - These algorithms include a work factor that increases the time complexity of the calculation
  - scrypt also requires a large amount of memory to compute, further complicating brute-force attacks
  - Increase in authentication time is negligible for user

Password Storage Summary

1. **Never store passwords in plain text**
2. **Password recovery from hashed passwords can be done with dictionary attacks (common passwords) and pre-computed rainbow tables (uncommon passwords)**
3. **Always salt and hash passwords before storing them**
4. **Use hash functions with a high work factor (bcrypt or scrypt)**

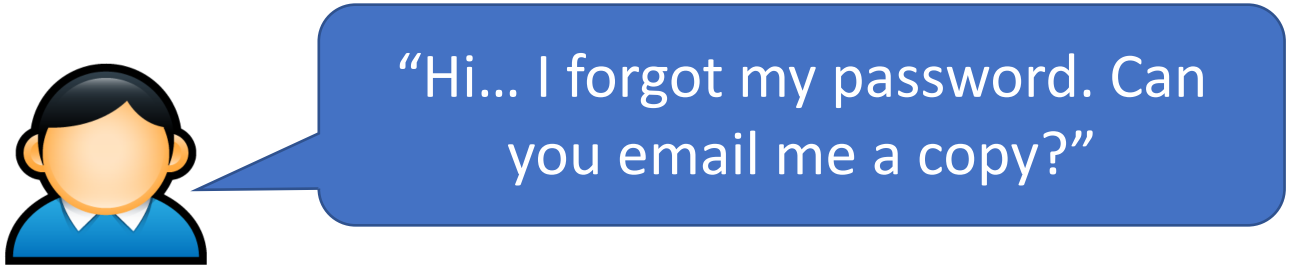
- These rules apply to any system that needs to authenticate users
  - Operating systems, websites, phones, etc.

Password Authentication Lifecycle Diagram



Password Recovery/Reset

- Problem: hashed passwords cannot be recovered (hopefully)



- This is why systems typically implement password reset
  - Use out-of-band info to authenticate the user
  - Overwrite hash(old_pw) with hash(new_pw)
- Be careful: its possible to crack password reset

Cracking Password Reset

- Typical implementations use Knowledge Based Authentication (KBA)
  - What was your mother's maiden name?
  - What was your prior street address?
  - Where did you go to elementary school?
- Problems?
  - This information is widely available to anyone
  - Publicly accessible social network profiles
  - Background-check services like Spokeo
- Experts recommend that services not use KBA
  - When asked, users should generate random answers to these questions

Choosing Passwords

Bad Algorithms

Better Heuristics

Password Reuse

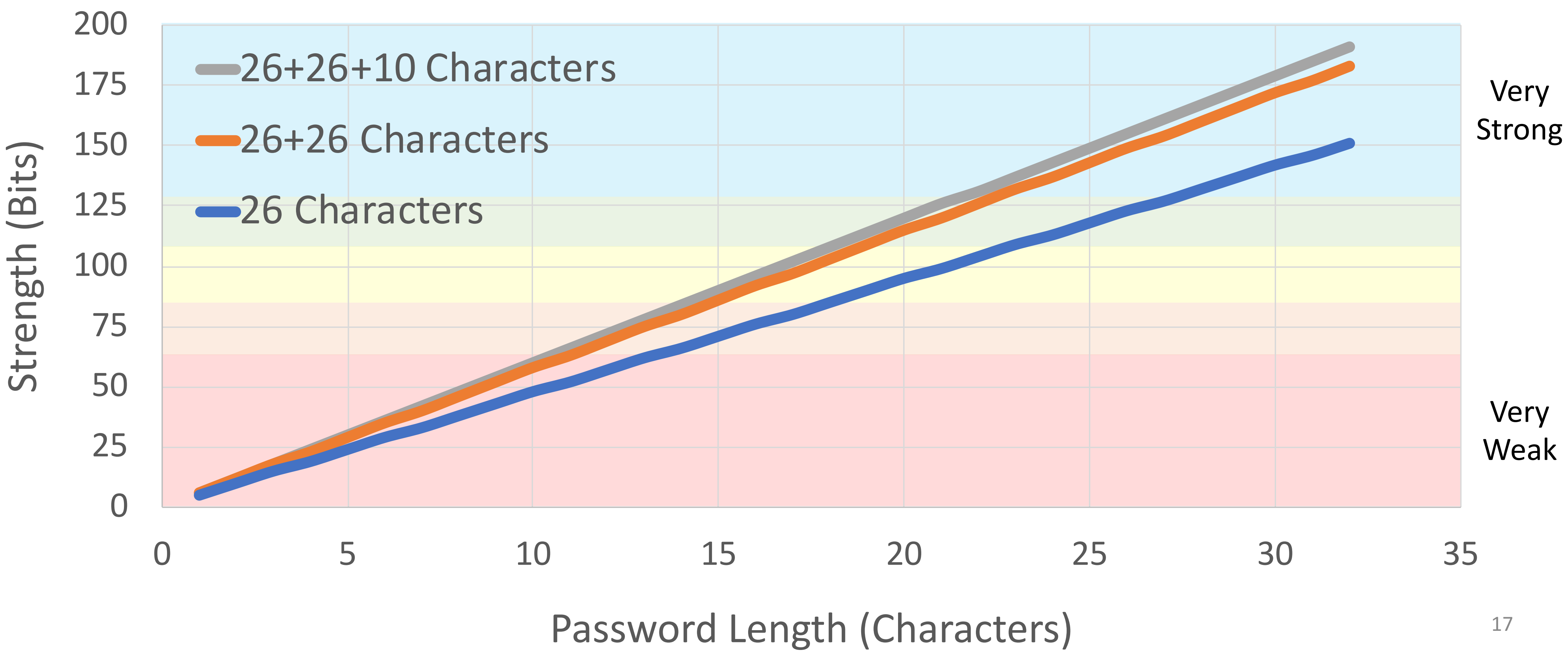
Password Quality

$$S = log_2 N^L \rightarrow L = \frac{S}{log_2 N}$$

- How do we measure password quality? Entropy
  - *N* – the number of possible symbols (e.g. lowercase, uppercase, numbers, etc.)
  - *L* – the length of the password
  - *S* – the strength of the password, in bits
- Formula tells you length *L* needed to achieve a desired strength S…
  - … for **randomly generated** passwords
- Is this a realistic measure in practice?

The Strength of Random Passwords

$$S = L * log_2 N$$



Mental Algorithms

- Years of security advice have trained people to generate “secure” passwords

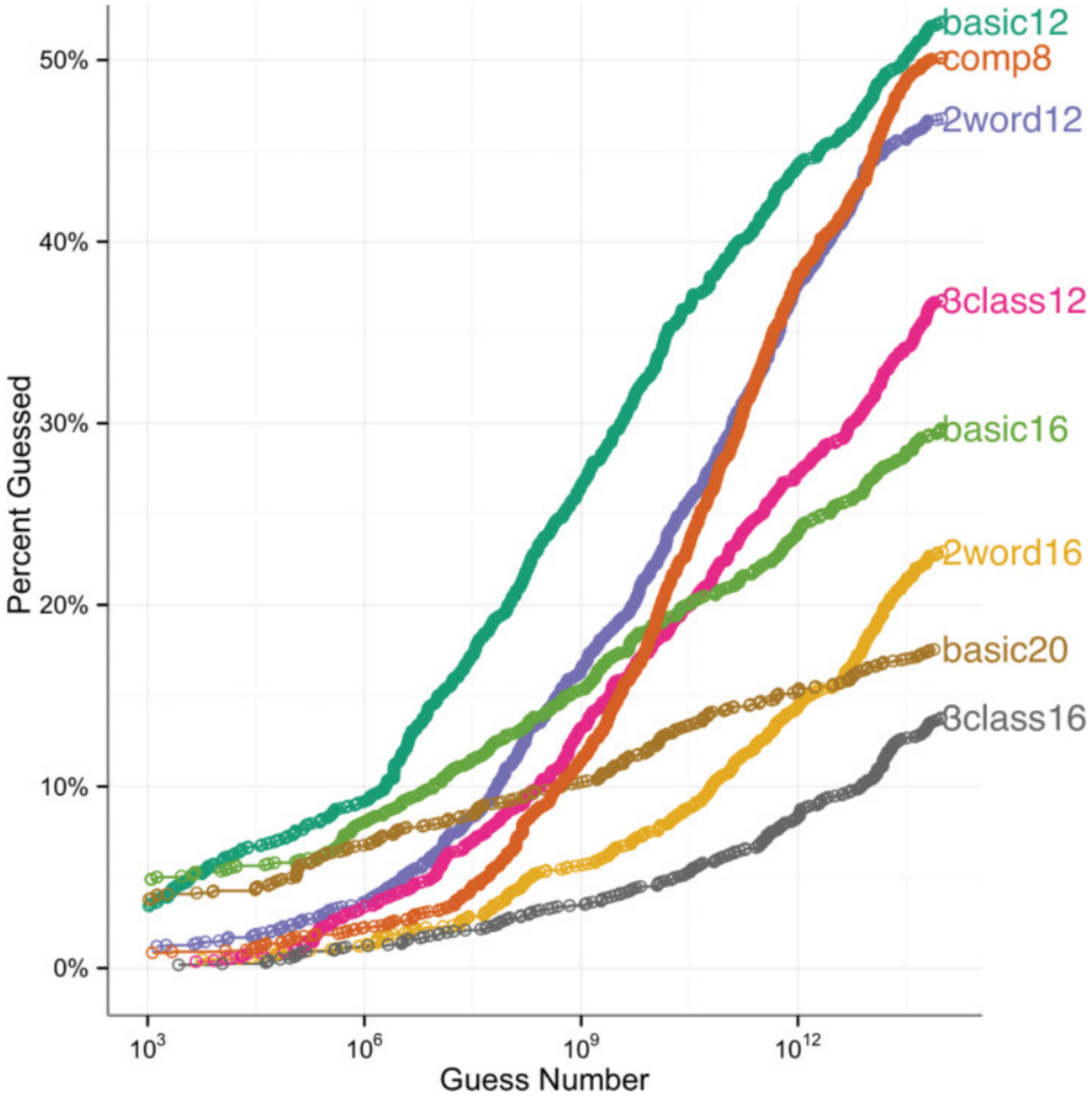
1. Pick a word
2. Capitalize the first or last letter
3. Add a number (and maybe a symbol) to the beginning or end

1. Pick a word
2. Replace some of the letters with symbols (a → @, s → $, etc.)
3. Maybe capitalize the first or last letter

Human Generated Passwords

Password	Entropy (bits)	Strength	Crackability	Problem
Computer3@	60	Weak	Easy	Dictionary word, obvious transformations
cl4ssr00m	47	Weak	Easy	Dictionary word, obvious transformations
7Dogsled*	54	Weak	Easy	Dictionary word, obvious transformations
Tjw1989&6	54	Weak	Easy	Users initials and birthday, obvious transformations
B4nk0f4m3r1c4!	83	Medium	Easy	Includes service name, obvious transformations

- Modern attackers are sophisticated
  - No need for brute force cracking!
  - Use dictionaries containing common words and passwords from prior leaks
  - Apply common “mental” permutations



Password Requirements

- comp *n* and basic *n*: use at least *n* characters
- *k* word *n*: combine at least *k* words using at least *n* characters
- *d* class *n*: use at least *d* character types (upper, lower, digit, symbol) with at least *n* characters

Plot from Shay et al.
https://www.blaseur.com/papers/tissec_1026.pdf

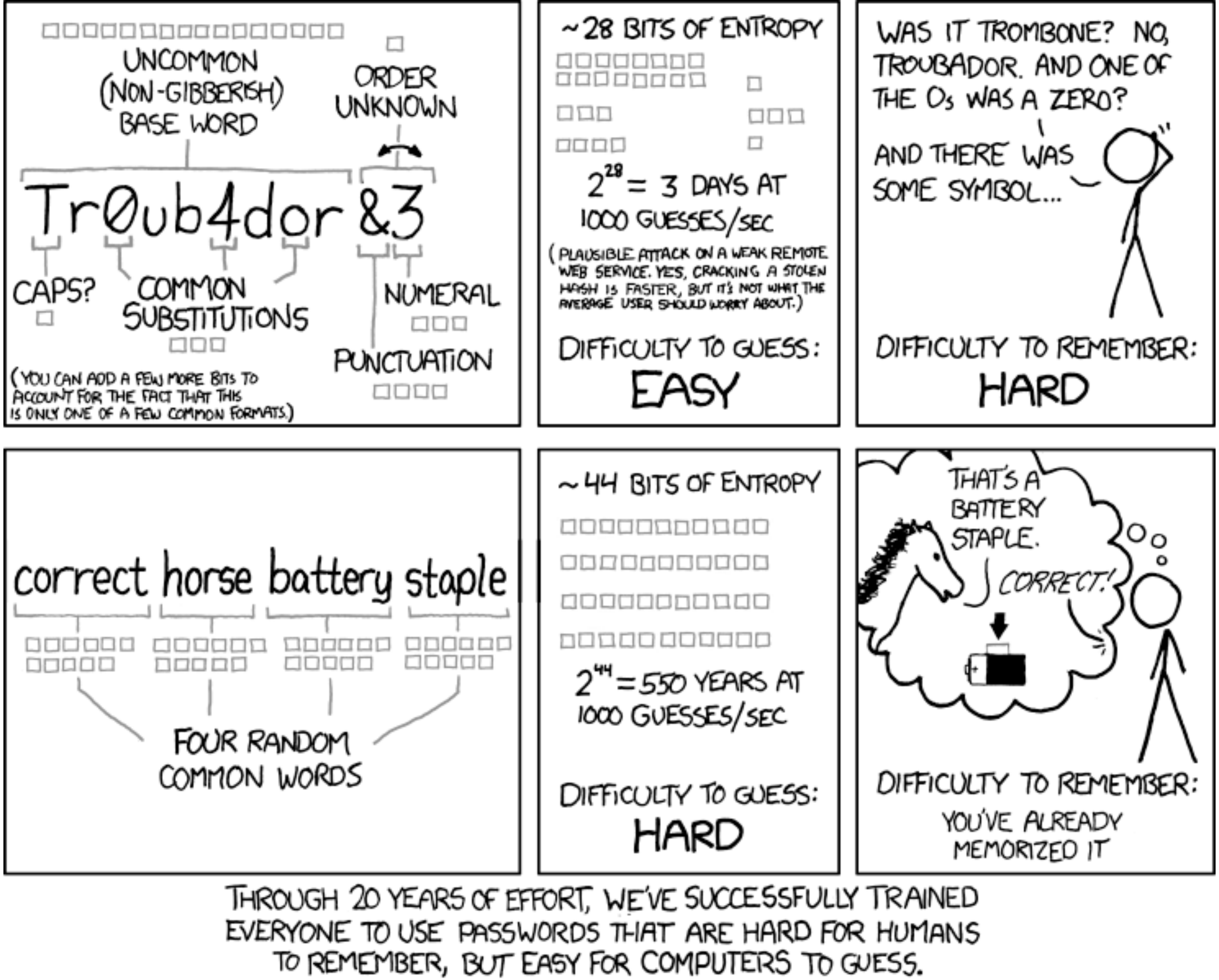
Better Heuristics

- Notice that in $S = L * log_2 N$, length matters more than symbol types
  - Choose longer passwords (16+ characters)
  - Don't worry about uppercase, digits, or symbols

- Use mnemonics
  - Choose a sentence or phrase
  - Reduce it to the first letter of each word
  - Insert random uppercase, digits, and symbols

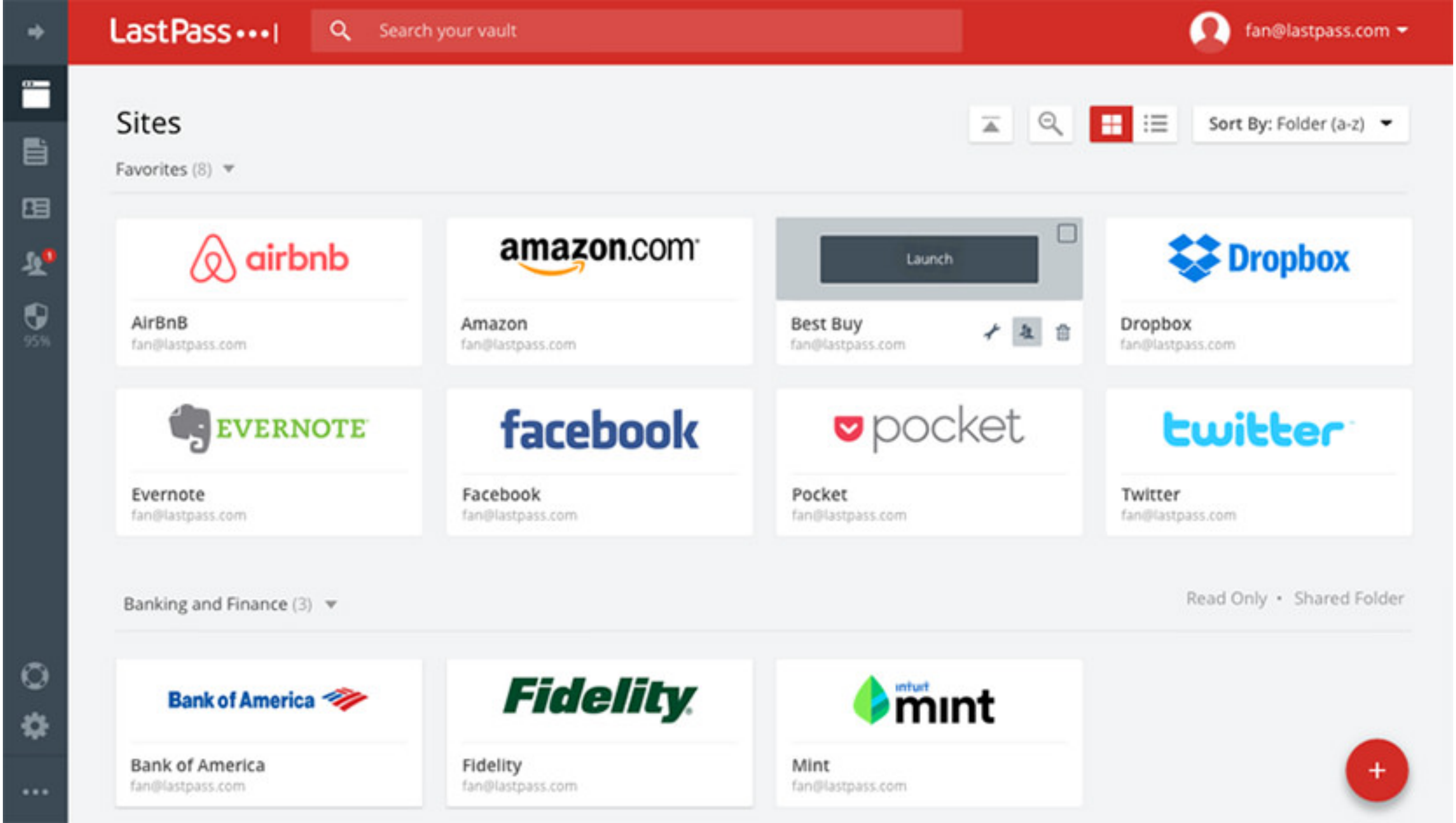
I double dare you, say "what" one more time

i2Dy,s"w"omt


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
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.

Password Reuse

- People have difficulty remembering >4 passwords
  - Thus, people tend to reuse passwords across services
  - What happens if any one of these services is compromised?
- Service-specific passwords are a beneficial form of compartmentalization
  - Limits the damage when one service is inevitably breached
- Use a password manager
- Some service providers now check for password reuse
  - Forbid users from selecting passwords that have appeared in leaks


LastPass •••|
Search your vault
fan@lastpass.com
95%
Sites
Sort By: Folder (a-z)
Favorites (8)
airbnb
AirBnB
fan@lastpass.com
amazon.com
Amazon
fan@lastpass.com
Launch
Best Buy
fan@lastpass.com
Dropbox
Dropbox
fan@lastpass.com
EVERNOTE
Evernote
fan@lastpass.com
facebook
Facebook
fan@lastpass.com
pocket
Pocket
fan@lastpass.com
twitter
Twitter
fan@lastpass.com
Banking and Finance (3)
Read Only • Shared Folder
Bank of America
Bank of America
fan@lastpass.com
Fidelity
Fidelity
fan@lastpass.com
intuit mint
Mint
fan@lastpass.com

Two Factor Authentication

Biometrics

SMS

Authentication Codes

Smartcards & Hardware Tokens

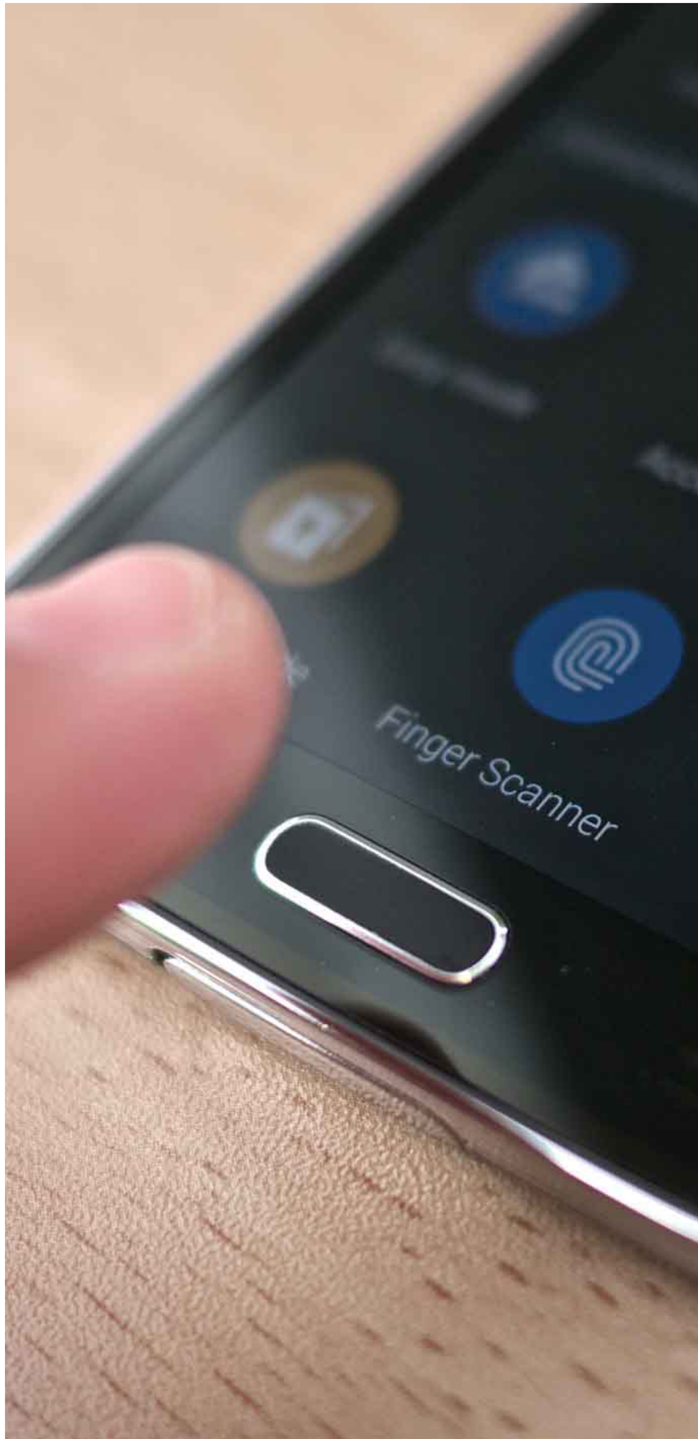
Biometrics

- Ancient Greek: bios ="life", metron ="measure“
- Physical features
  - Fingerprints
  - Face recognition
  - Retinal and iris scans
  - Hand geometry
- Behavioral characteristics
  - Handwriting recognition
  - Voice recognition
  - Typing cadence
  - Gait

Fingerprints

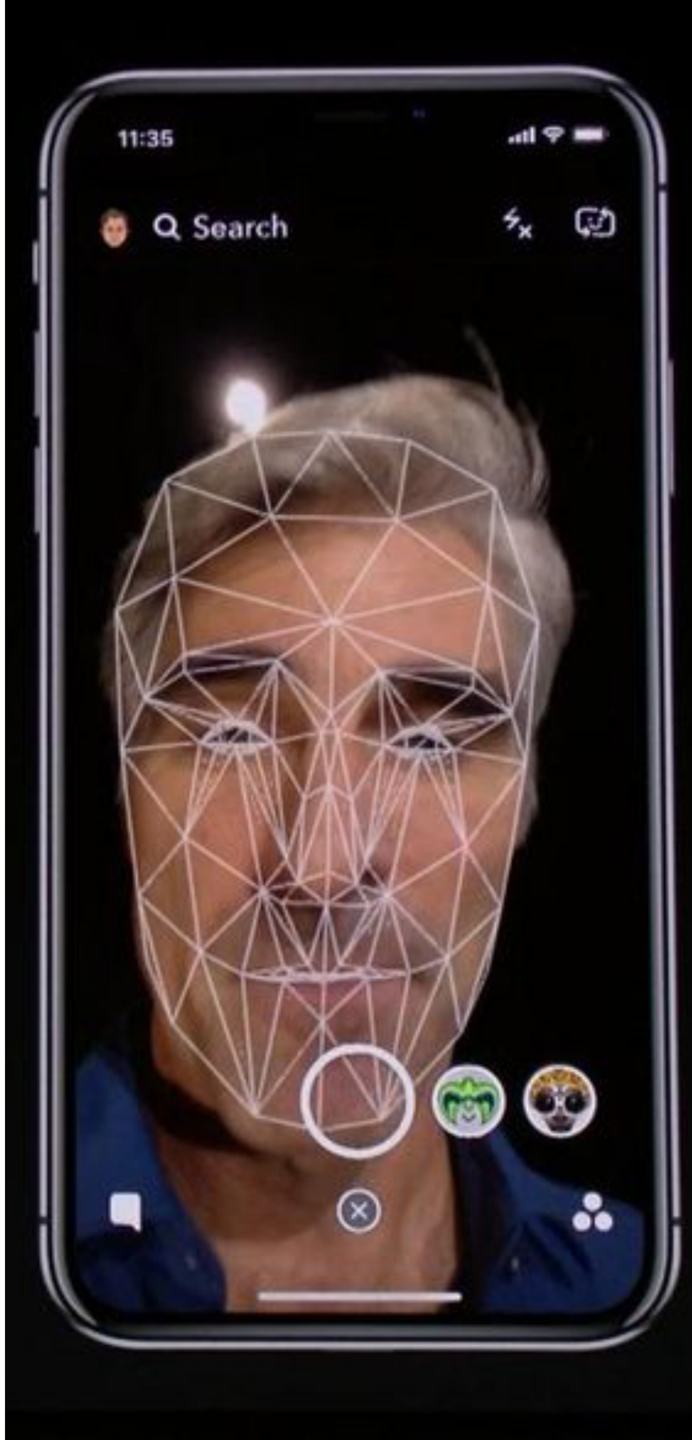
- Ubiquitous on modern smartphones, some laptops
- Secure?
  - May be subpoenaed by law enforcement
  - Relatively easy to compromise
    1. Pick up a latent fingerprint (e.g. off a glass) using tape or glue
    2. Photograph and enhance the fingerprint
    3. Etch the print into gelatin backed by a conductor
    4. Profit ;)

https://www.theregister.co.uk/2002/05/16/gummi_bears_defeat_fingerprint_sensors/



Facial Recognition

- Popularized by FaceID on the iPhone X
- Secure?
  - It depends
- Vulnerable to law enforcement requests
- Using 2D images?
  - Not secure
  - Trivial to break with a photo of the target's face
- Using 2D images + 3D depth maps?
  - More secure, but not perfect
  - Can be broken by crafting a lifelike mask of the target



Fundamental Issue With Biometrics

- Biometrics are immutable
  - You are the password, and you can’t change
  - Unless you plan on undergoing plastic surgery?
- Once compromised, there is no reset
  - Passwords and tokens can be changed
- Example: the Office of Personnel Management (OPM) breach
  - US gov agency responsible for background checks
  - Had fingerprint records of all people with security clearance
  - Breached by China in 2015, all records stolen :(

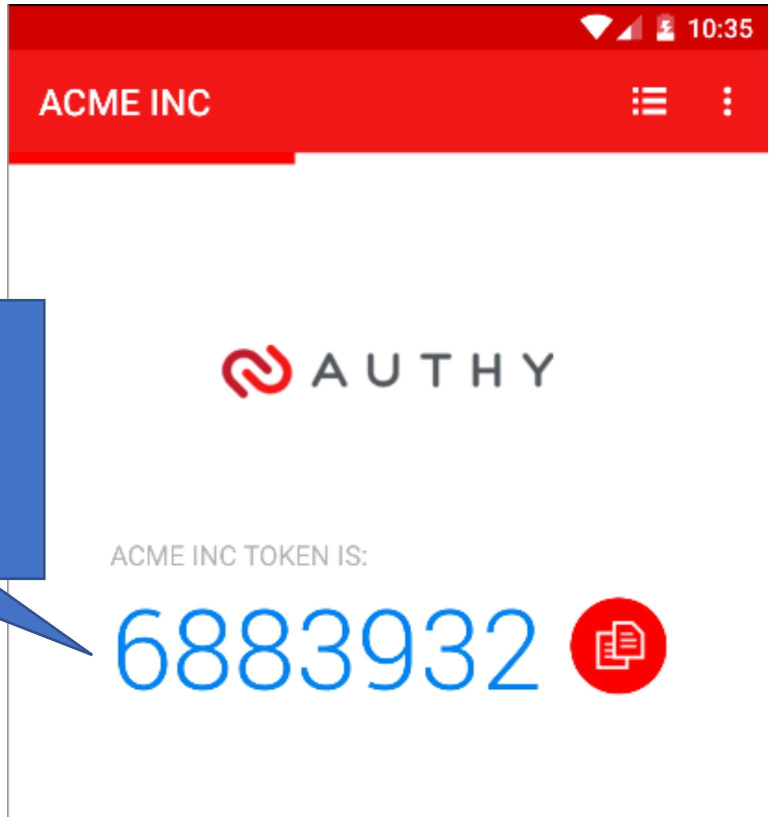
Something You Have

- Two-factor authentication has become more commonplace
- Possible second factors:
  - SMS passcodes
  - Time-based one time passwords
  - Hardware tokens

One Time Passwords

- Generate ephemeral passcodes that change over time
- To login, supply normal password and the current one time password
- Relies on a shared secret between your mobile device and the service provider
    - Shared secret allows both parties to know the current one time password
    - Every time period, the one time password is computed from shared secret and current time
        - Symmetric-key crypto

Changes every few minutes



Duo Mobile

Lastpass Authenticator

Google Authenticator

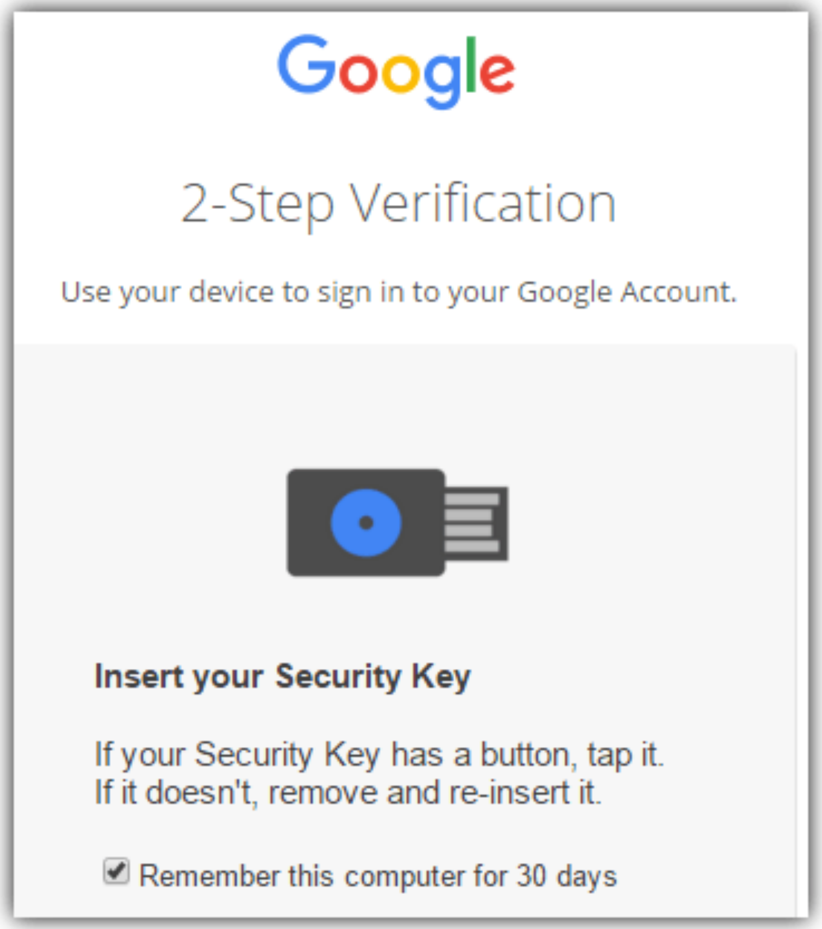
Hardware Two Factor

- Special hardware designed to hold cryptographic keys
- Physically resistant to key extraction attacks
  - E.g. scanning tunneling electron microscopes
- Uses:
  - 2^{nd} factor for OS log-on
  - 2^{nd} factor for some online services
  - Storage of PGP and SSH keys

Universal 2nd Factor (U2F)

- Supported by Chrome, Opera, and Firefox (must be manually enabled)
- Works with Google, Dropbox, Facebook, Github,
- Public key registered with website (site specific)
  - Secret keys are stored on device
- Generate signature to prove presence of device
  - Signature computation on device
  - How to prevent replaying an old signature?
  - Use a nonce (random number) sent by web site

- Pro tip: always buy 2 security keys
  - Associate both with your accounts
  - Keep one locked in a safe, in case you lose key ;)



Password Threat Modeling

Attacker	Action	Mitigations	Assumptions
Active: get access to user account	Perform dictionary attacks on passwd files	Strong passwords Salt and hash	Cannot compute enough hashes
Active	Dedicated hardware for faster hashing	Key stretching (slower hashing)	Cannot perform enough computation
Active	Attack password recovery	Life questions	Attacker cannot guess them
Active	Get access to user password or attack password recovery	One-time password	Attacker cannot guess one-time code Secret key not compromised
Active	Get access to user password or attack password recovery	Universal second factors	Cannot replay old authentication messages

Authentication Protocols

Unix, PAM

Kerberos

Status Check

- At this point, we have discussed:
  - How to securely store passwords
  - Techniques used by attackers to crack passwords
  - Biometrics and 2nd factors
- Next topic: building authentication systems
  - Given a user and password, how does the system authenticate the user?
  - How can we perform efficient, secure authentication in a distributed system?

Authentication in Unix/Linux

- Users authenticate with the system by interacting with *login*
  - Prompts for username and password
  - Credentials checked against locally stored credentials
- By default, password policies specified in a centralized, modular way
  - On Linux, using Pluggable Authentication Modules (PAM)
  - Authorizes users, as well as environment, shell, prints MOTD, etc.

Example PAM Configuration

```
# cat /etc/pam.d/system-auth
#%PAM-1.0

auth required pam_unix.so try_first_pa
auth optional pam_permit.so
auth required pam_env.so

account required pam_unix.so
account optional pam_permit.so
account required pam_time.so

password required pam_unix.so try_first_pass nullok sha512 shadow
password optional pam_permit.so

session required pam_limits.so
session required pam_unix.so
session optional pam_permit.so
```

- Use SHA512 as the hash function
- Use /etc/shadow for storage

Unix Passwords

- Traditional method: *crypt*
  - 25 iterations of DES on a zeroed vector
  - First eight bytes of password used as key (additional bytes are ignored)
  - 12-bit salt
- Modern version of *crypt* are more extensible
  - Support for additional hash functions like MD5, SHA256, and SHA512
  - Key lengthening: defaults to 5000 iterations, up to $10^8 - 1$
  - Full password used
  - Up to 16 bytes of salt

Password Files

- Password hashes used to be in */etc/passwd*
  - World readable, contained usernames, password hashes, config information
  - Many programs read config info from the file…
  - But very few (only one?) need the password hashes
- Turns out, world-readable hashes are **Bad Idea**
- Hashes now located in */etc/shadow*
  - Also includes account metadata like expiration
  - Only visible to root

Password Storage on Linux

/etc/passwd

username:x:UID:GID:full_name:home_directory:shell

charlie:x:1001:1000:Charlie S:/home/charlie/:/bin/bash
home/alice/:/bin/sh

$<algo>$<salt>$<hash>
Algo: 1 = MD5, 5 = SHA256, 6 = SHA512

etc/shadow

rname:password:last:may:must:warn:expire:disable:reserved

charlie:$1$0nSd5ewF$0df/3G7iSV49nsbAa/5gSg:9479:0:10000::::
alice:1l3RxU5F1$:8172:0:10000::::

Distributed Authentication

- Early on, people recognized the need for authentication in distributed environments
  - Example: university lab with many workstations
  - Example: file server that accepts remote connections
- Synchronizing and managing password files on each machine is not scalable
  - Ideally, you want a centralized repository that stores policy and credentials

Kerberos

- Created as part of MIT Project Athena
  - Based on Needham-Schroeder
- Provides mutual authentication over untrusted networks
  - Tickets as assertions of authenticity, authorization
  - Forms basis of Active Directory authentication
- Principals
  - Client
  - Server
  - Key distribution center (KDC)
    - Ticket granting server (TGS)