Authentication
Part 1

Blase Ur (and David Cash and Grant Ho)
May 12, 2026
CMSC 23200

THE UNIVERSITY OF CHICAGO

Who Am I?

- Grant Ho
  - Distinguished security researcher
  - Recently moved here from California; hates the cold
  - Fan of hot dogs
  - Canvas hacking expert

Or Am I?

How (and why) do we authenticate users?

Authentication in the Abstract

- **Principal:** legitimate owner of an identity
- **Claimant:** entity trying to be authenticated
- Verify that **people** or **things** (e.g., server) are who they claim, or maybe that the claimant has some **attribute**
- Authentication ≠ Authorization ≠ Access Control
  - *Authorization* is defining whether an authenticated entity should have access to a given resource or take some action
  - *Access control* is the enforcement of authorization policies

Authentication Use Cases

- Explicit authentication
  - Single-factor authentication
  - Multi-factor authentication (e.g., with Duo)
- Implicit authentication
  - Continuous authentication (e.g., with behavioral biometrics)
- Risk-based authentication: vary auth requirements based on estimated risk

How We Authenticate (1/2)

- Something you know
  - Password
  - PIN (Personal Identification Number)
- Something you have
  - Private key (of a public-private key pair)
  - Hardware device (often with a key/seed)
  - Phone (running particular software)
  - Token (e.g., string stored in a cookie)
  - Access to an email account / phone (code, magic link)

How We Authenticate (2/2)

- Something you are
  - Biometrics (e.g., iris or fingerprint)
  - Behavioral tendencies (behavioral biometrics)
- Somewhere you are
  - IP address
  - Location-limited channels
- Some system or person vouches for you
  - Single sign-on (e.g., UChicago Okta, Google, Facebook)
  - PKI Certificate Authorities

123456789
password
iloveyou
princess
rockyou
nicole
liverpool
12345678
tigger
shadow
purple
michael
melissa
summer
christian
princess1
sakura
soccer
flower
jessica
andrea
tinkerbell
junior
ashley
password1
daniel
1234567890
anthony
joseph
jesus
mickey
loveme
elizabeth
brandon
poohbear
basketball
butterfly
superman
miguel
matthew
softball
lovely
jordan
football
playboy
666666
jasmine
qwerty
angels
justin
andrew
secret
pretty
barbie
chelsea
taylor
danielle
hello
abc123
patrick

Why Are Passwords So Prevalent?

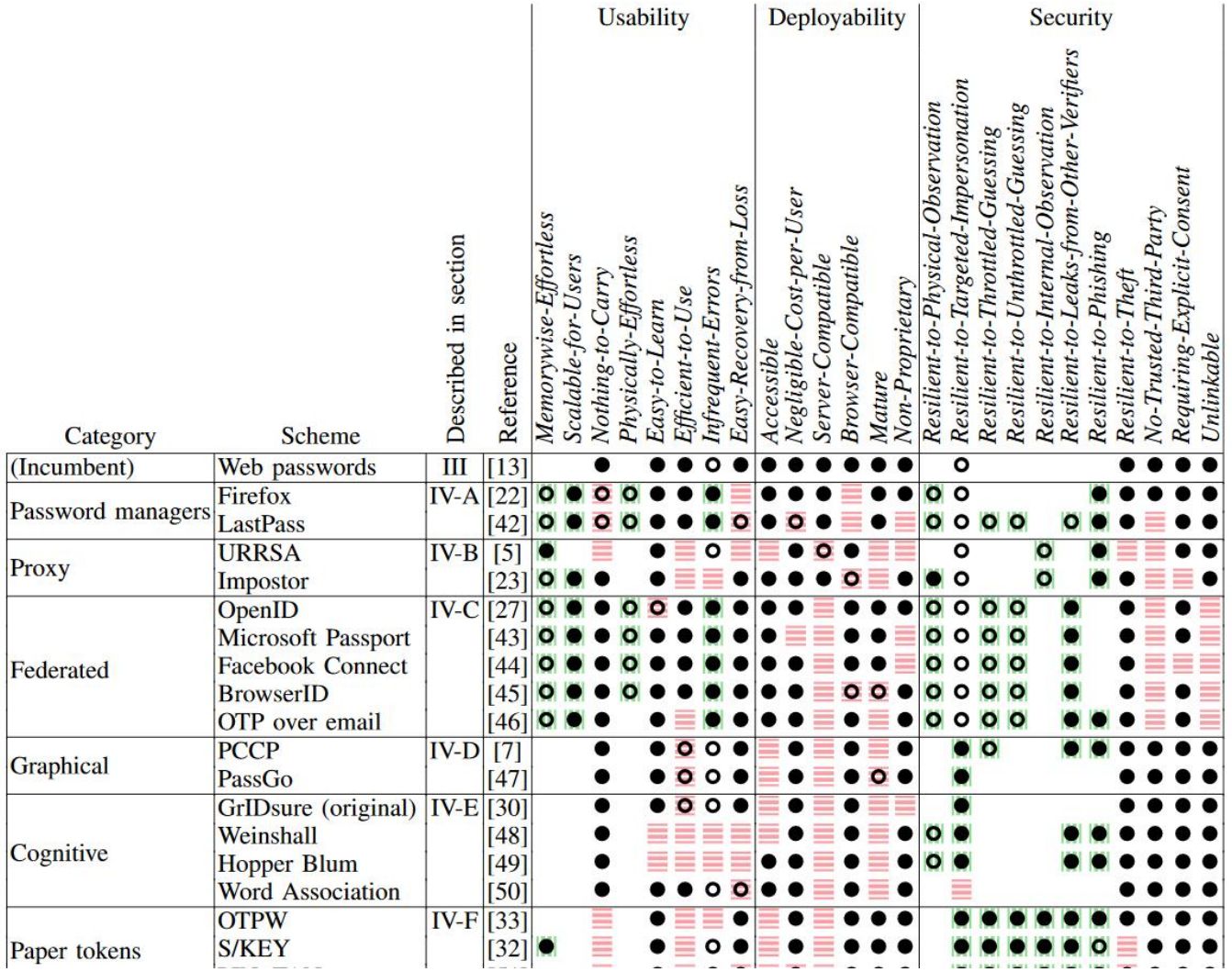
- Easy to use
- Easy to deploy
- Nothing to carry
- No “silver-bullet” alternative

Why Are Passwords So Prevalent?

Memorywise-Effortless	Usability
Scalable-for-Users	
Nothing-to-Carry	
Physically-Effortless	
Easy-to-Learn	
Efficient-to-Use	
Infrequent-Errors	
Easy-Recovery-from-Loss	
Accessible	Deployability
Negligible-Cost-per-User	
Server-Compatible	
Browser-Compatible	
Mature	
Non-Proprietary	
Resilient-to-Physical-Observation	Security
Resilient-to-Targeted-Impersonation	
Resilient-to-Throttled-Guessing	
Resilient-to-Unthrottled-Guessing	
Resilient-to-Internal-Observation	
Resilient-to-Leaks-from-Other-Verifiers	
Resilient-to-Phishing	
Resilient-to-Theft	
No-Trusted-Third-Party	
Requiring-Explicit-Consent	
Unlinkable	

Bonneau et al. “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes,” In *Proc. IEEE S&P*, 2012

Why Are Passwords So Prevalent?



Category	Scheme	Described in section	Reference	Usability								Deployability						Security											
				Memorywise-Effortless	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*
(Incumbent)	Web passwords	III	[13]	●	○	●		●	●	○	●	●	●	●	●	●	●		○						●	●	●	●
Password managers	Firefox	IV-A	[22]	○	●	○	○	●	●	●	▬	●	●	●	▬	●	▬	○	○					●	●	●	●	●	●
	LastPass		[42]	○	●	○	○	●	●	●	○	●	○	●	▬	●	▬	○	○	○	○		○	●	●	▬	●	●
Proxy	URRSA	IV-B	[5]	●		▬		●	▬	○	▬	▬	●	○	●	▬	▬		○			○		●	▬	▬	●	●
	Impostor		[23]	○	●	●		●	▬	▬	●	●	●	●	○	▬	●	●	○			○		●	●	▬	▬	●	●
Federated	OpenID	IV-C	[27]	○	●	●	●	○	○	●	●	●	●	▬	●	●	●	○	○	○	○		●		●	▬	●	▬
	Microsoft Passport		[43]	○	●	●	●	○	●	●	●	●	▬	▬	●	●	▬	○	○	○	○		●		●	▬	●	▬
	Facebook Connect		[44]	○	●	●	●	○	●	●	●	●	●	▬	●	●	▬	○	○	○	○		●		●	▬	▬	▬
	BrowserID		[45]	○	●	●	●	○	●	●	●	●	●	▬	○	○	●	○	○	○	○		●		●	▬	●	▬
	OTP over email		[46]	○	●	●		●	▬	●	●	●	●	▬	●	▬	●	○	○	○	○		●	●	●	▬	●	▬
Graphical	PCCP	IV-D	[7]			●		●	○	○	●	▬	●	▬	●	▬	●		●	○			●	●	●	●	●	●
	PassGo		[47]			●		●	○	○	●	▬	●	▬	●	○	●		●				●	●	●	●	●	●
Cognitive	GrIDsure (original)	IV-E	[30]			●		●	○	○	●	▬	●	▬	●	▬	▬		●						●	●	●	●
	Weinshall		[48]			●		▬	▬	▬	▬	▬	●	▬	●	▬	●	○	●				●	●	●	●	●	●
	Hopper Blum		[49]			●		▬	▬	▬	▬	●	●	▬	●	▬	●	○	●				●	●	●	●	●	●
	Word Association		[50]			●		●	●	○	○	●	●	▬	●	▬	▬		▬						●	●	●	●
Paper tokens	OTPW	IV-F	[33]			▬		●	▬	▬	●	▬	●	▬	●	●	▬		●	●	●	●	●	●	●	●	●	●
	S/KEY		[32]	●		▬		●	▬	○	●	▬	●	▬	●	●	▬		●	●	●	●	●	○	▬	●	●	●

Bonneau et al. “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes,” In *Proc. IEEE S&P*, 2012

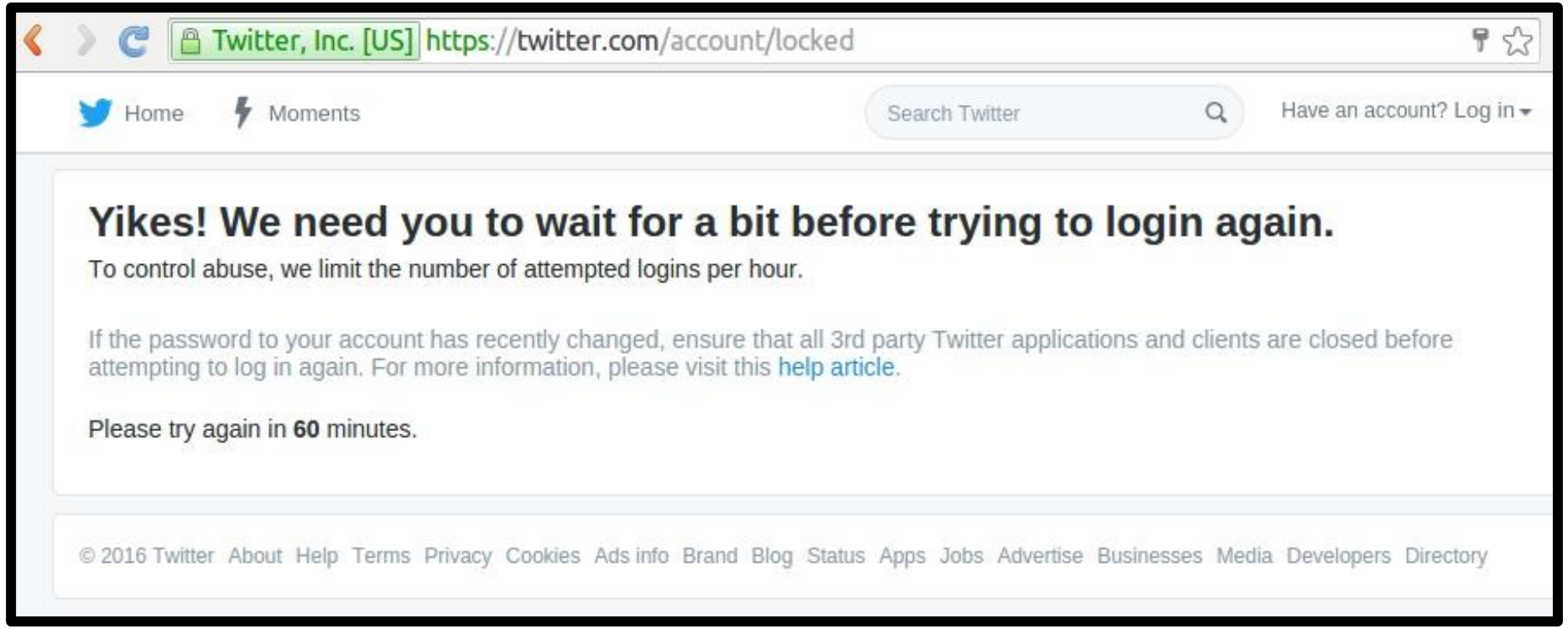
Threat Modeling For Passwords (Discussion)

One Guessing Attack Against Passwords

- **Online attack**
  - Try passwords on a live system (online)
  - Try to authenticate to a device (e.g., smartphone)

Key Defense Against Online Attacks

- Rate limiting
- On hardware devices, usually requires secure hardware



Another Guessing Attack Against Passwords

- **Offline attack** (web)
  - Hacker steals password database (e.g., using SQL injection)
  - Try to guess passwords in the password database
    - Detour: Why do we need to guess passwords? Don't we just have them?
- Attacking a file encrypted using a key derived from a password (e.g., with PBKDF2) is similar

Defending Against Offline Attacks

- Thought experiment: Why don’t we just keep the passwords on an air-gapped system so that the attacker can’t get to them?

Defending Against Offline Attacks

- Thought experiment: Why don’t we encrypt passwords so that if the attacker steals the password database, they only have the encrypted passwords?

Detour: Storing Passwords

- Goal: Prevent attacker from being able to use a stolen password database immediately (without more work)
- Hash function: one-way function
  - Traditionally designed for efficiency (e.g., MD5, SHA-2), but don't ever use those!
  - Use password-specific functions (e.g., bcrypt, scrypt, Argon2)

Offline Attack (In Practice)

- Attacker compromises database (e.g., via SQL injection)
  - md5("blase") = `12b872adb2588c668d706d847fc1da7e`
  - Attacker makes guesses (from most likely/probable to the least) and hashes those guesses
  - Attack is limited only by the computational power the attacker will invest
- Attacker model: untargeted ("trawling") vs. targeted
- Finds match → try on this site, try on other sites
  - Password **reuse** is a core problem (more next lecture!)

Hashing on one NVIDIA RTX 4090

- Hashcat benchmarks
- MD5: ~ 150 billion / second
- SHA-1: ~ 50 billion / second
- UNIX md5crypt: ~ 60 million / second
- NTLM: ~ 250 billion / second
- SHA-2 (256): ~ 20 billion / second
- bcrypt (32 iterations): ~ 240,000 / second
- scrypt (16,384 iterations): ~ 7,000 / second

https://www.onlinehashcrack.com/tools-benchmark-hashcat-nvidia-rtx-4090.php

Storing Passwords

- **Salt**: random string assigned per-user
  - Combine the password with the salt, then hash it
  - Stored alongside the hashed password

Generic hash types

Hash-Mode	Hash-Name	Example
0	MD5	8743b52063cd84097a65d1633f5c74f5
10	md5($pass.$salt)	01dfae6e5d4d90d9892622325959afbe:7050461
20	md5($salt.$pass)	f0fda58630310a6dd91a7d8f0a4ceda2:4225637426

Bcrypt (password-specific hash function) example:

`$2a$04$iHdEgkI681VdDMc3f7edau9phRwORvhYjqWAIb7hb4B5uFJO1g4zi`

$ = delimiter

2a = bcrypt

04 = 2^4 iterations (cost)

iHdEgkI681VdDMc3f7edau = 16 bytes of salt (radix-64 encoded)

9phRwORvhYjqWAIb7hb4B5uFJO1g4zi = 24 bytes of hash output (radix-64 encoded)

Storing Passwords

- Highly advisable practice: both **salt** and **hash** passwords
  - Prevents the use of rainbow tables (hash outputs precomputed for many passwords, mapping sorted by *output*)
  - Increases the attacker's work proportional to the number of accounts

Typical (Web) Account Creation

- User sends **username** and desired **password** over an encrypted tunnel
- Server validates username (e.g., does it exist in the system?) and password (e.g., does it meet composition requirements?)
- Server generates a random salt
  - Think about how long the salt should be!
- Server stores **username**, **salt**, and **hash(password|salt)** in database

Typical (Web) Authentication

- User sends **username** and **password$_0$** over an encrypted tunnel
- Server looks up the salt and hash output associated with that username
- Server computes hash(password$_0$|salt)
- If the computed value matches the hash output in the database, typically send back auth token (long string attacker can't guess associated with that user's session) and store it as a "Secure=True" (HTTPS-only) cookie

Offline Attack Revisited

- Attacker model for an offline attack: attacker generates likely candidates, hashes those candidates (adding the salt if applicable), and keeps guessing until they give up
- Attacking a file encrypted using a key derived from a password (e.g., with PBKDF2) is similar to an offline attack trying to guess hashes

Other Attacks Against Passwords

- **Shoulder surfing**: looking at someone else entering their credentials

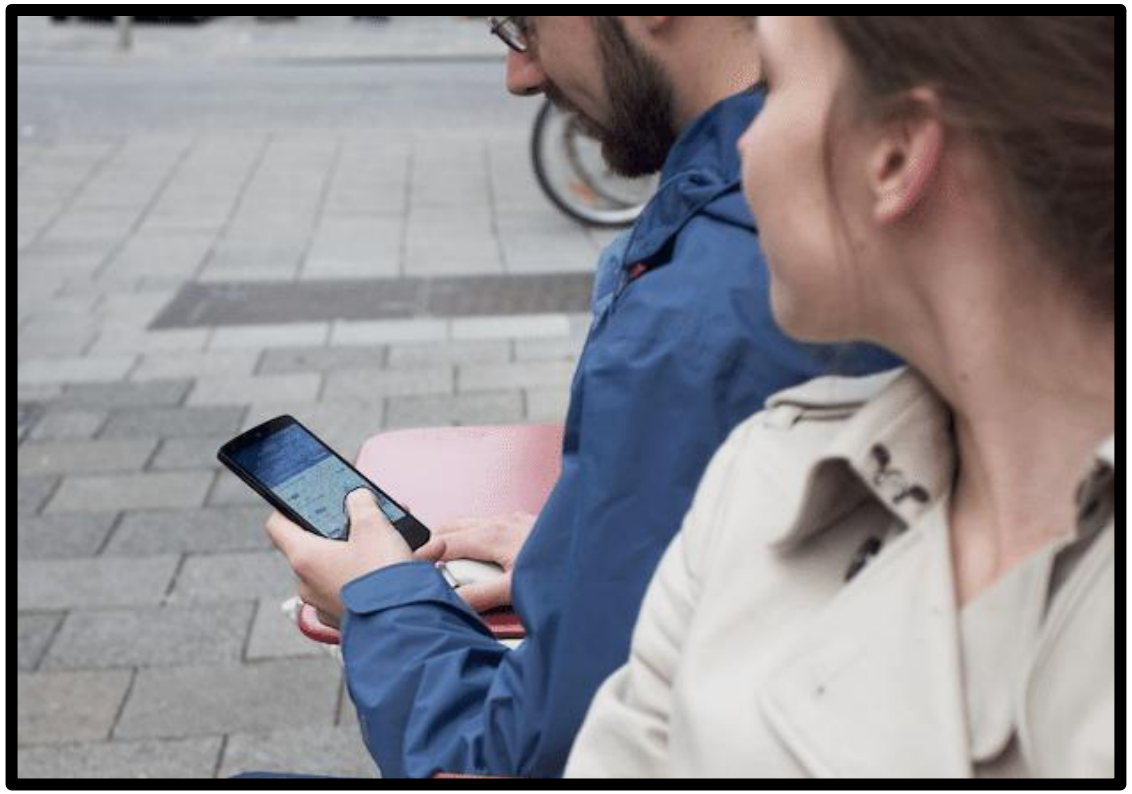
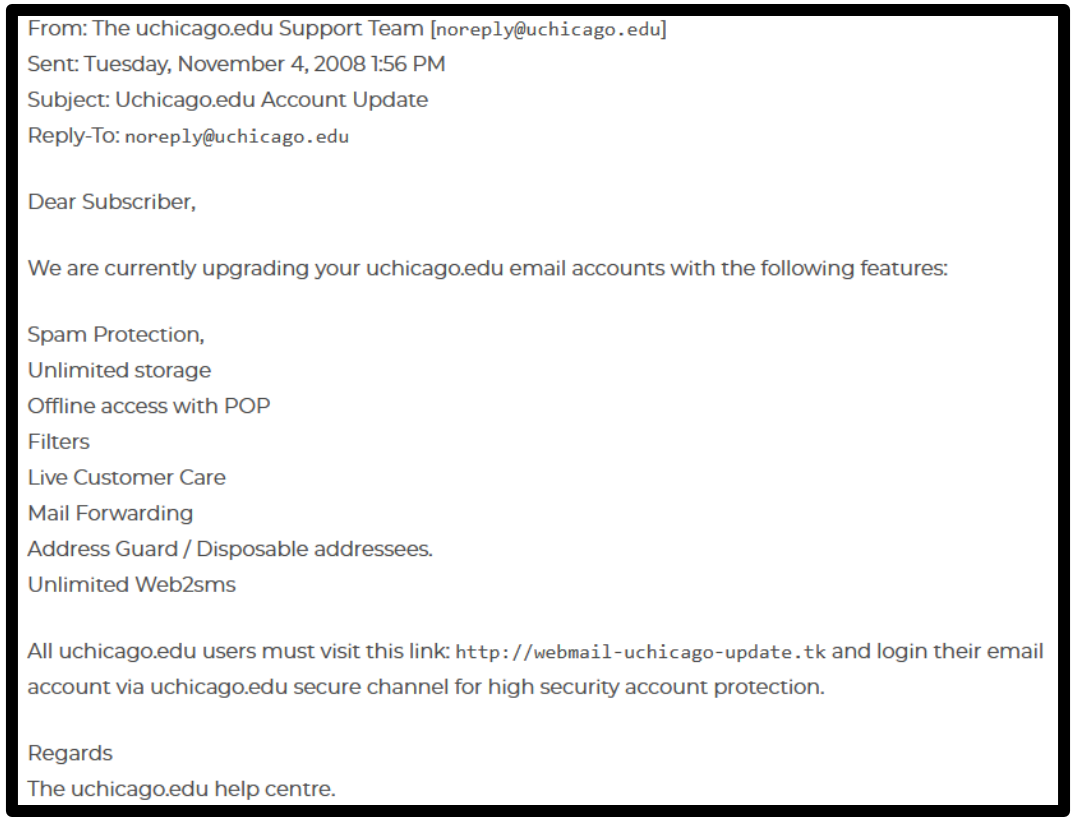


Photo from https://www.researchgate.net/figure/A-shoulder-surfing-situation-in-a-cafe_fig1_312490451

Other Attacks Against Passwords

- **Phishing** attack: try to trick the user into giving their credentials to you, believing you are the legitimate system



From: The uchicago.edu Support Team [noreply@uchicago.edu]
Sent: Tuesday, November 4, 2008 1:56 PM
Subject: Uchicago.edu Account Update
Reply-To: noreply@uchicago.edu

Dear Subscriber,

We are currently upgrading your uchicago.edu email accounts with the following features:

Spam Protection,
Unlimited storage
Offline access with POP
Filters
Live Customer Care
Mail Forwarding
Address Guard / Disposable addressees.
Unlimited Web2sms

All uchicago.edu users must visit this link: http://webmail-uchicago-update.tk and login their email account via uchicago.edu secure channel for high security account protection.

Regards
The uchicago.edu help centre.

Example from https://intranet.uchicago.edu/tools-and-resources/it-services/security/phishing/examples-of-phishing-scams

Other Attacks Against Passwords

- **Spear phishing**: a phishing attack targeted to the recipient (e.g., using their personal data)

Credential Breaches

Data-Driven Statistical Attacks

- (2009) 32 million passwords: 

  – In plaintext!

Data-Driven Statistical Attacks

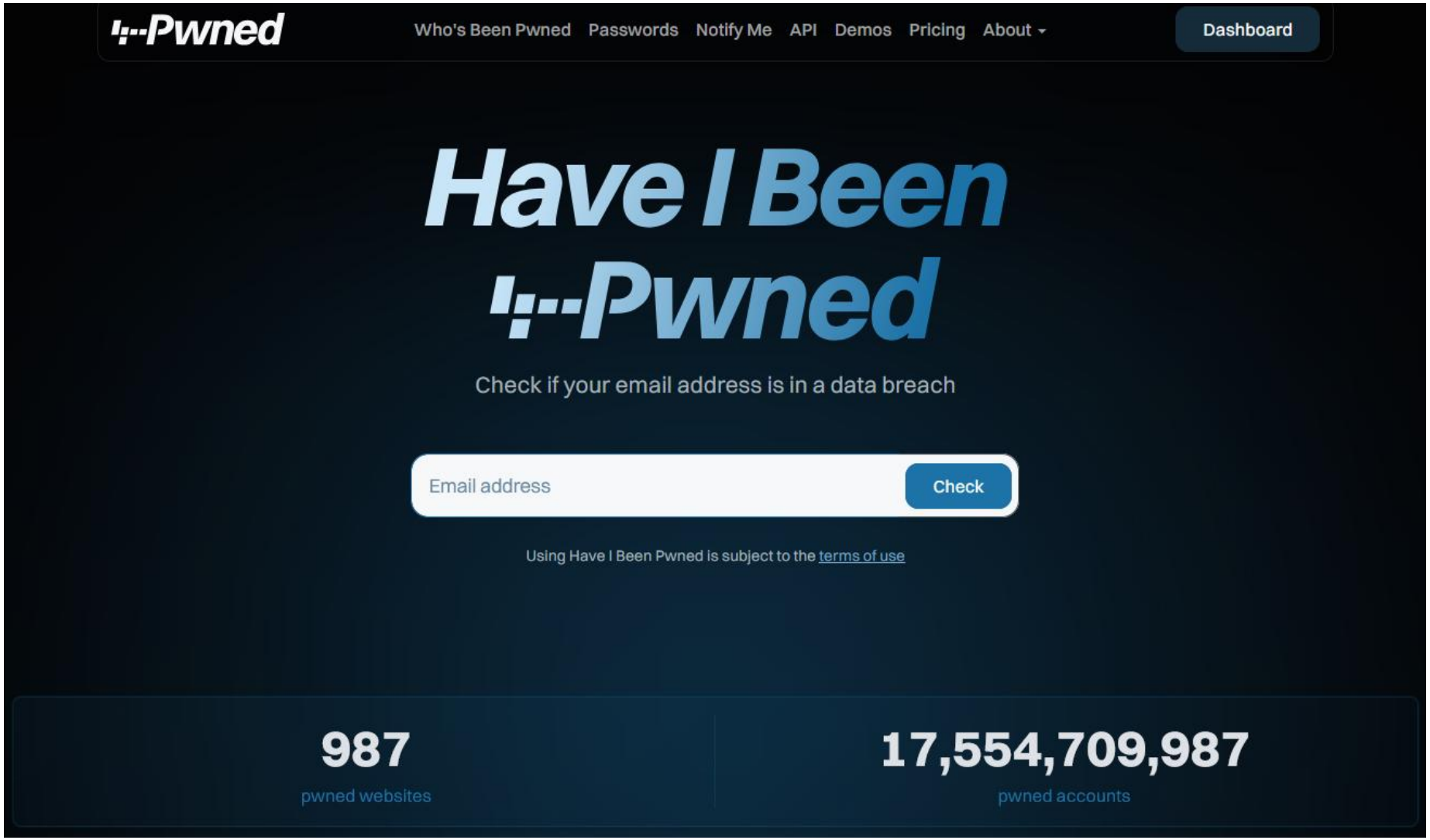
- (2009) 32 million passwords: 
  - In plaintext!

- (2016, but from 2012) 164 million passwords: 
  - Unsalted SHA-1

Data-Driven Statistical Attacks

- (2009) 32 million passwords: rockyou
  - In plaintext!
- (2016, but from 2012) 164 million passwords: LinkedIn
  - Unsalted SHA-1
- (2017) 3 billion passwords: YAHOO!
  - Still not released publicly as of 2026

Have I Been Pwned (as of 5/12/26)



User Modeling: Password Selection

- Most secure option: assign password to users
- More realistic option: let users pick a password
  - Have some policy around what passwords are allowed
  - Password-composition policy: minimum requirements (and sometimes maximum requirements for legacy systems)
  - Blocklist = disallowed passwords

What Do Human-Chosen Passwords Look Like? (Live Demo)