User Authentication

- Continues to be challenging because security isn't just technical
  - Users pick bad passwords
- Model:
  - Client sends request to server
  - How does the server check each request for authentication?
  - How do they implement a **guard**
- Three parts:
  - Registration: set up secret
  - Authentication: check user sent secret to ensure it matches with copy
  - **Recovery**: what happens if user loses secret
- Principal = user that wants to be authenticated
- Challenge: intermediaries
  - The request from the user goes through many intermediaries
  - Final server just receives a TCP packet
  - Need to consider the possibility that the request isn't actually from the user
    - I.e. attacker controls load balancer / user device / etc.
- Challenge: verifying user's identity on registration
  - How do we know the person who registered is actually who they say they are?
  - Some companies don't care (i.e. Amazon) as long as you just pay

Registration

- Approaches:
  - First-come first-serve:
    - Register for an account at gmail.com
  - Bootstrap from another mechanism (i.e. email)
  - Created by administrator
    - I.e. for new employees

Recovery

- Approaches:
  - Security questions
  - Verify via email
  - Prove knowledge of credit card number
  - Create new account
  - Call customer service (subsceptible to social engineering)

Passwords

- Commonly used as a secret
- Principal and guard share a secret set of bits
- Easy to use and deploy, but often very weak
  - Use them as little as possible (just for user authentication)
  - Once a user is authenticated, use crypto keys between server / clients
  - Even for user authentication, can combine with other ideas such as password manager; single-sign on, two-factor, etc.
- Problems with apsswords:
  - Users choose guessable passwords
    - 20% of accounts use the same set of 5,000 most popular passwords

- Common passwords contain digits / upper case / lower case
  - What matters is entropy: 1 bit of entropy = password would be guessed on the first attempt half the time
  - Password of 16 bits of entropy requires 2^16 guesses to try all possibilities
- Passwords often shared across sites / apps
- Want to encourage users to choose high-entropy passwords
  - Idea is that even if adversaries obtain old password, they're no longer useful
  - But users might have a harder time remembering it

Password Managers

- Automates selecting different password with high entropy for different sights
- User must authenticate to this and memorize one strong password

Defense Against Guessing

- Guessing attacks are a problem in a small key space
  - Adversary has access to common passwords / phrases / common user biases
- Data encrypted with solely a password is vulnerable to offline guesses if the attacker doesn't have to go through a server each time

---

- Strategies:
  - Limiting authentic attempts
    - Rate-limit login attempts and time-out periods after too many incorrect guesses
    - Limiting per-user might not be enough because attacker could instead guess "1234" for every username
    - CAPTCHA can help, but the cost of solving them nowadays is quite low
    - Most systems have heuristics to rate-limit

Storing Passwords

- Shouldn't store passwords directly
- Rainbow tables are dictionary of hashes of all common passwords
  - Can be used to break if you just store hash of password
- Hashing with salting can be much more effective
- Hash functions should be purposefully expensive / slow (key derivation function) to rate-limit more effectively
  - I.e. bcrypt, scrypt, PBKDF2

2FA

- Defends against MITM and phishign attacks
- Variants
  - SMS
    - Easy to use / recover
    - Requires user to trust cell phone network / be in range of cell phone network and still enables phishing attacks
    - Compromising the server does not leak secrets (just a cell-phone numeber)
  - TOTP
    - Time-based one-time passwords
    - Server and user device agree on secret value and then generate code based on hash of secret and current time
    - Server checks if that code is correct by also looking aththe current time
    - No need for cell phone network anymore, but can be difficult with user setup, changing devices, re-registering secrets if server is compromised

        - Still allows phishing attacks
    - Challenge-response (U2F)
        - New section

U2F

- User's USB dongle has a public/secret key pair
    - Server stores public key
    - To log in, server sends random challenge to user borwser
    - USB dongle signs this challenge with private key
    - This can be sent to server for authentication
- This disables phishing attacks / compromising server vulnerabilities but is unwieldy for the user
- Problem: how does this prevent phishing attacks if attacker visits the site and then forwards this to the user?
    - The challenge sent by the server should be tied to the user's identity
    - I.e. the nonce could contain a hash of the requestor's IP / port / hostname
    - This way, a user could only sign the nonce if they see that it was actually meant for them
    - If attacker tried to ask server, they would get a nonce with the attacker's details
    - Also have the U2F signing protocol sign not just the challenge but instead `challenge | server details`
        - I.e. append the challenge to the server hostname / port / etc.
- New problem: different servers can see which of their users match up by just comparing the public keys
    - Problem since we normally cannot do this with passwords!
    - Known as cross-site linking
    - At registration time, device generates new key pair `(Kp, Ks)`
    - It has a secret `Kw` that it wraps around `Ks` and then sends to the site it is registering with
    - When a site wants to get a login, it sends the wrapped `Ks` and the `Kp`
        - The device can then decrypt `Kp` and get this working