Authentication

For a user (or process):
• Get the user’s identity = identification
• Verify the identity = authentication
• Then decide whether to allow access to resources = authorization

Three Factors of Authentication

1. Ownership
   • Key, card
   • Can be stolen
2. Knowledge: something you know
   • Passwords, PINs
   • Can be guessed, shared, stolen
3. Inherence: something you are
   • Biometrics
   • Usually needs hardware, can be copied (sometimes)
   • Once copied, you’re stuck

Multi-Factor Authentication

Factors may be combined
– ATM machine: 2-factor authentication
   • ATM card something you have
   • PIN something you know
– Password + code delivered via SMS: 2-factor authentication
   • Password something you know
   • Code validates that you possess your phone

Two passwords ≠ Two-factor authentication

Authentication: PAP

Password Authentication Protocol

• Unencrypted, reusable passwords
• Insecure on an open network
• Also, password file must be protected from open access
   • But administrators can still see everyone’s passwords

PAP: Reusable passwords

PROBLEM 1: Open access to the password file
What if the password file isn’t sufficiently protected and an intruder gets hold of it? All passwords are now compromised!
Even if a trusted admin sees your password, this might also be your password on other systems.

Solution:
Store a hash of the password in a file
– Given a file, you don’t get the passwords
– Attacker must resort to a dictionary or brute-force attack
– Example, Linux passwords are hashed with SHA-512 hashes (SHA-2)

PROBLEM 2: Sniffing
Someone who can see network traffic (or over your shoulder) can see the password!
Authentication: CHAP

Challenge-Handshake Authentication Protocol

The challenge is a nonce (random bits).
We create a hash of the nonce and the secret.
An intruder does not have the secret and cannot do this!

CHAP authentication

Alice network host

"alice" look up alice's key, K

generate random challenge number C

R' = f(K, C)

"welcome"

R = R' ⊕?

an eavesdropper does not see K

Time-Based Authentication

Time-based One-time Password (TOTP) algorithm

- Both sides share a secret key
- User runs TOTP function to generate a one-time password
  
  one_time_password = hash(secret_key, time)

- User logs in with:
  
  Name, password, and one_time_password

- Service generates the same password
  
  one_time_password = hash(secret_key, time)

Public Key Authentication

Public key authentication

Demonstrate we can encrypt or decrypt a nonce
This shows we know the key

- Alice wants to authenticate herself to Bob:
  
  - Bob generates nonce, S
  - Sends it to Alice
  
  - Alice: encrypts S with her private key (signs it)
  - Sends result to Bob

For mutual authentication, Alice must present Bob with a nonce that Bob will encrypt with his private key and return
Public Keys as Identities

A public key can be treated as an identity
- Only the owner of the corresponding private key will be able to create the signature
- New identities can be created by generating new random \{private, public\} key pairs

Anonymous identity – no identity management
- A user is known by a random-looking public key
- Anybody can create a new identity at any time
- Anybody can create as many identities as they want
- A user can throw away an identity when it is no longer needed
- Example: Bitcoin identity = hash(public key)

Public key authentication

- Public key authentication relies on binding identity to a public key
  - How do you know it really is Alice’s public key?
- One option: Get keys from a trusted source
  - Problem: requires always going to the source
    - Cannot pass keys around
- Another option: sign the public key
  - Contents cannot be modified
  - digital certificate

X.509 Certificates

ISO introduced a set of authentication protocols
X.509: Structure for public key certificates:
- Subject
  - Distinguished name
- Public key
  - (algorithm & key)
- Version
- Serial #
- Issuer
  - Distinguished Name
- Validity
  - (from-to)
- Signature
  - Algorithm
  - (signed by CA)

Reminder: What’s a digital signature?

Hash of a message encrypted with the signer’s private key

Transport Layer Security (TLS)

Certificates prevent someone from using a phony public key to masquerade as another person
…but if you trust the CA
Transport Layer Security (TLS)

Goal: provide a transport layer security protocol
After setup, applications feel like they are using TCP sockets

SSL: Secure Socket Layer

Created with HTTP in mind
- Web sessions should be secure
- Mutual authentication is usually not needed
  - Client needs to identify the server, but the server won’t know all clients
  - Rely on passwords after the secure channel is set up
 Enables TCP services to engage in secure, authenticated transfers
- http, telnet, rftp, ftp, smtp, xmpp, ... 

SSL evolved to TLS (Transport Layer Security)

TLS Protocol

Goal: Provide authentication (usually one-way), privacy, & data integrity between two applications

- Principles
  - Authentication
    - Use public key cryptography & X.509 certificates for authentication
    - Optional – can authenticate 0, 1, or both parties
  - Data encryption
    - Use symmetric cryptography to encrypt data
    - Key exchange: random keys generated at the start of each session
      Diffie-Hellman or public key
  - Data integrity
    - Include a MAC with transmitted data to ensure message integrity
  - Interoperability & evolution
    - Support many different key exchange, encryption, integrity, & authentication protocols – negotiate what to use at the start of a session

TLS Protocol

(1) Client hello
   Version & crypto information
(2) Server hello
   Server certificate
   [client certificate request]
(3) Verify server certificate
(4) Client key exchange
   Send encrypted session key
   [ (5) Send client certificate ]
(6) Verify server certificate
(7) Client done
(8) Server done
(9) Communicate
Symmetric encryption + MAC

OAuth 2.0

OAuth 2.0: Open Authorization

OAuth: framework for service authorization
- Allows you to authorize one website (consumer) to access data from another website (provider) – in a restricted manner
- Designed initially for web services
- Examples:
  - Allow the Moo photo printing service to get photos from your Flickr account
  - Allow the NY Times to tweet a message from your Twitter account

OpenID Connect
- Remote identification: use one login for multiple sites
- Encapsulated within OAuth 2.0 protocol
OAuth setup

OAuth is based on
- Getting a token from the service provider & presenting it each time an application accesses an API at the service
- URL redirection
- JSON data encapsulation

Before users can use OAuth, the app (consumer) must register with the service provider
- Service provider (e.g., Flickr):
  - Gets data about your application (name, creator, URL)
  - Assigns the application (consumer) an ID & a secret
    - ID = unique ID for the app (consumer); secret = shared secret between app and service provider
  - Presents list of authorization URLs and scopes (access types)

How does authorization take place?

Application needs an Access Token from the Service (e.g., moo.com needs an access token from flickr.com)
- Application redirects user to Service Provider
  - Request contains: client ID, client secret, scope (list of requested APIs)
  - User may need to authenticate at that provider
  - User authorizes the requested access
  - Service Provider redirects back to consumer with a one-time-use authorization code
- Application now has the Authorization Code
  - The previous redirect passed the Authorization Code as part of the HTTP request – therefore not encrypted
- Application exchanges Authorization Code for Access Token
  - The legitimate app uses HTTPS (encrypted channel) & sends its secret
  - The application now talks securely & directly to the Service Provider
  - Service Provider returns Access Token
- Application makes API requests to Service Provider using the Access Token

OAuth Entities

REDIRECT: client ID, return-URL, scope (list of requested APIs)

Moo.com app redirects you to the service provider

You authenticate (optional) & authorize the request at flickr

You want moo.com to access your photos on flickr

You want moo.com to access your photos on flickr

You want moo.com to access your photos on flickr
Flicker sends a redirect back with an authorization code.

Moo requests an access token (securely).

Moo requests an access token (securely).

- You may still need to log into the Provider’s OAuth service when redirected.
- You approve the specific access that you are granting.
- The Service Provider validates the requested access when it gets a token from the Consumer.

Play with it at the OAuth 2.0 Playground: https://developers.google.com/oauthplayground/
Single Sign-On: OpenID Connect

- Designed to solve the problem of
  - Having to get an ID per service (website)
  - Managing passwords per site
- Layer on top of OAuth 2.0
- Decentralized mechanism for single sign-on
  - Access different services (sites) using the same identity
    - Simplify account creation at new sites
    - User chooses which OpenID provider to use
    - Website never sees your password
- OpenID Connect is a standard but not the only solution
  - Used by Google, Microsoft, Amazon Web Services, PayPal, Salesforce, ...
  - Facebook Connect – popular alternative solution
    (similar in operation but websites can share info with Facebook, offer friend access, or make suggestions to users based on Facebook data)

OpenID Connect Authentication

- OAuth requests that you specify a "scope"
  - List of access methods that the app needs permission to use
- To enable user identification:
  - Specify "openid" as a requested scope
- Send request to the identity provider
  - Handles user authentication
  - Redirects the user back to the client
- Provider returns an access token and an ID token
  - The access token contains:
    - approved scopes
    - expiration
    - etc.
  - The ID token can be read by the consumer (client) and contains
    - Name, screen name, email, birthdate, ... whatever the Identity Provider chose to send

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- Hash functions
- Random number generators
  - Used for nonces and session keys

Examples

- Key exchange
  - Public key cryptography
- Key exchange + secure communication
  - Random # + public key cryptography + symmetric cryptography
- Authentication
  - Nonce (random #) + encryption
- Message authentication code
  - Hash + symmetric cryptography
- Digital signature
  - Hash + public key cryptography

The End