

Distributed Systems

25. Authentication

Paul Krzyzanowski
Rutgers University
Fall 2018

November 26, 2018 © 2018 Paul Krzyzanowski 1

Authentication

- For a user (or process):
 - Establish & verify identity
 - Then decide whether to allow access to resources (= authorization)

November 26, 2018 © 2017-2017 Paul Krzyzanowski 2

Authentication

Three factors:

- something you have *key, card*
 - Can be stolen
- something you know *passwords*
 - Can be guessed, shared, stolen
- something you are *biometrics*
 - Usually needs hardware, can be copied (sometimes)
 - Once copied, you're stuck

November 26, 2018 © 2017-2017 Paul Krzyzanowski 3

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

November 26, 2018 © 2017-2017 Paul Krzyzanowski 4

Authentication: PAP

Password Authentication Protocol

```

    graph LR
      client[client] -- "login, password" --> server[server]
      server -- "OK" --> client
      subgraph server_db [server]
        direction TB
        db["name: password database"]
      end
  
```

- 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

November 26, 2018 © 2017-2017 Paul Krzyzanowski 5

PAP: Reusable passwords

PROBLEM: 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
- Have to resort to a **dictionary** or **brute-force attack**
- Example, passwords hashed with SHA-512 hashes (SHA-2)

November 26, 2018 © 2017-2017 Paul Krzyzanowski 6

What is salt?

- How to speed up a dictionary attack
 - Create a table of **precomputed hashes**
 - Search(hashed_password) → original_password
- How to stop dictionary attacks
 - Salt** = random string (typically up to 16 characters)
 - Concatenated with the password
 - Stored with the password file (it's not secret)
 - Even if you know the salt, you cannot use precomputed hashes to search for a password (because the salt is prefixed)
 - Makes a table of precomputed hashes prohibitively huge

November 26, 2018 © 2017-2017 Paul Krzyzanowski 7

Authentication: CHAP

Challenge-Handshake Authentication Protocol

```

    graph LR
        client[client]
        server[server]
        client -- "challenge ← = nonce" --> server
        server -- "hash(challenge, secret)" --> client
        client -- "OK" --> server
        client --- secret1[Has shared secret]
        server --- secret2[Has shared secret]
    
```

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!

November 26, 2018 © 2017-2017 Paul Krzyzanowski 8

CHAP authentication

```

    graph LR
        Alice[Alice]
        network[network]
        host[host]
        Alice -- "alice" --> network
        network -- "alice" --> host
        host -- "look up alice's key, K" --> host
        host -- "generate random challenge number C" --> host
        host -- "C" --> Alice
        Alice -- "R' = f(K, C)" --> host
        host -- "R = f(K, C)" --> host
        host -- "R = R'?" --> host
        Alice -- "welcome" --> host
        subgraph Note
            direction TB
            NoteText[an eavesdropper does not see K]
        end
    
```

November 26, 2018 © 2017-2017 Paul Krzyzanowski 9

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
 - $one_time_password = hash(secret_key, time)$

November 26, 2018 © 2017-2017 Paul Krzyzanowski 10

Public Key Authentication

November 26, 2018 © 2017-2017 Paul Krzyzanowski 11

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

November 26, 2018 © 2017-2017 Paul Krzyzanowski 12

Public key authentication

Bob:

1. Look up "alice" in a database of public keys
2. Decrypt the message from Alice using Alice's public key
3. If the result is S , then Bob is convinced he's talking with Alice

For **mutual authentication**, Alice has to present Bob with a nonce that Bob will encrypt with his private key and return

November 26, 2018 © 2017-2017 Paul Krzyzanowski 13

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**

November 26, 2018 © 2017-2017 Paul Krzyzanowski 14

X.509 Certificates

ISO introduced a set of authentication protocols
X.509: Structure for public key **certificates**:

Issuer = **Certification Authority (CA)**

X.509 v3 Digital Certificate

Name, organization, locality, state, country, etc.

November 26, 2018 © 2017-2017 Paul Krzyzanowski 15

Reminder: What's a digital signature?

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

November 26, 2018 © 2017-2017 Paul Krzyzanowski 16

X.509 certificates

When you get a certificate

- Verify its signature:
 - hash contents of certificate data
 - Decrypt CA's signature with CA's public key

Obtain CA's public key (certificate) from trusted source

Certificates prevent someone from using a phony public key to masquerade as another person
...if you trust the CA

November 26, 2018 © 2017-2017 Paul Krzyzanowski 17

SSL/TLS

November 26, 2018 © 2017-2017 Paul Krzyzanowski 18

Transport Layer Security

- 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
- SSL evolved to **TLS (Transport Layer Security)**
 - SSL 3.0 was the last version of SSL ... and is considered insecure
 - We use TLS now ... but often still call it SSL

November 26, 2018

© 2017-2017 Paul Krzyzanowski

19

Transport Layer Security (TLS)

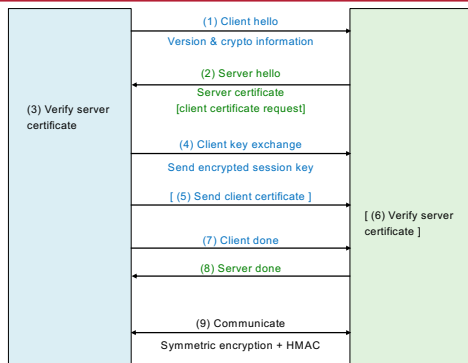
- aka **Secure Socket Layer (SSL)**, which is an older protocol
- Sits on top of TCP/IP
- Goal: provide an encrypted and possibly authenticated communication channel
 - Provides authentication via RSA and X.509 certificates
 - Encryption of communication session via a symmetric cipher
- **Hybrid cryptosystem**: (usually, but also supports Diffie-Hellman)
 - Public key for authentication
 - Symmetric for data communication
- Enables TCP services to engage in secure, authenticated transfers
 - http, telnet, ntp, ftp, smtp, ...

November 26, 2018

© 2017-2017 Paul Krzyzanowski

20

TLS Protocol



November 26, 2018

© 2017-2017 Paul Krzyzanowski

21

OAuth 2.0

November 26, 2018

© 2017-2017 Paul Krzyzanowski

22

Service Authorization

- You want an app to access your data at some service
 - E.g., access your Google calendar data
- But you want to:
 - Not reveal your password to the app
 - Restrict the data and operations available to the app
 - Be able to revoke the app's access to the data

November 26, 2018

© 2017-2017 Paul Krzyzanowski

23

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

November 26, 2018

© 2017-2017 Paul Krzyzanowski

24

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
- Register a service
 - Service provider (e.g., Flickr):
 - Gets data about your application (name, creator, URL)
 - Assigns the application (consumer) an ID & a secret
 - Presents list of authorization URLs and scopes (access types)

November 26, 2018 © 2017-2017 Paul Krzyzanowski 25

OAuth Entities

You want moo.com to access your photos on flickr

November 26, 2018 © 2017-2017 Paul Krzyzanowski 26

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**

November 26, 2018 © 2017-2017 Paul Krzyzanowski 27

OAuth Entities

You want moo.com to access your photos on flickr

November 26, 2018 © 2017-2017 Paul Krzyzanowski 28

OAuth Entities

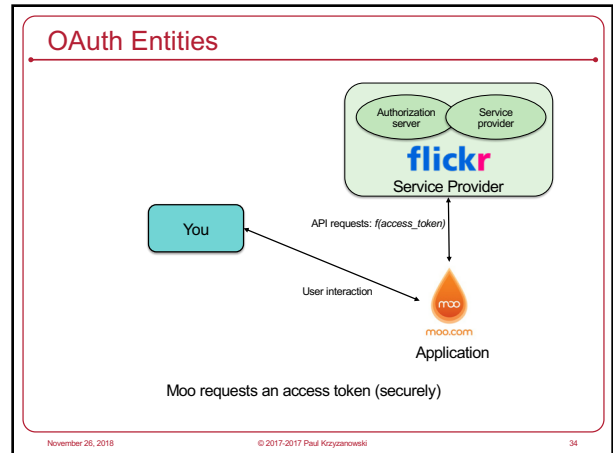
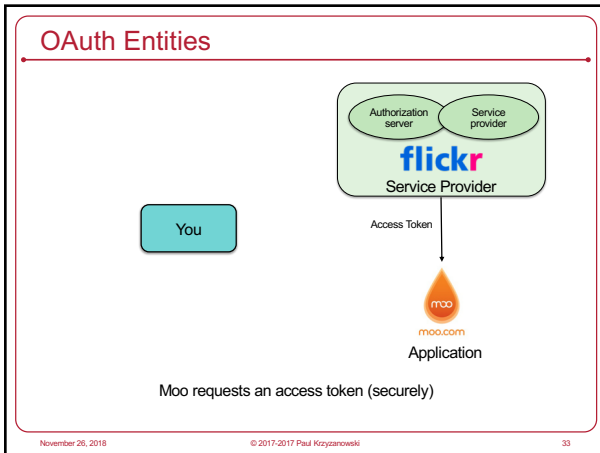
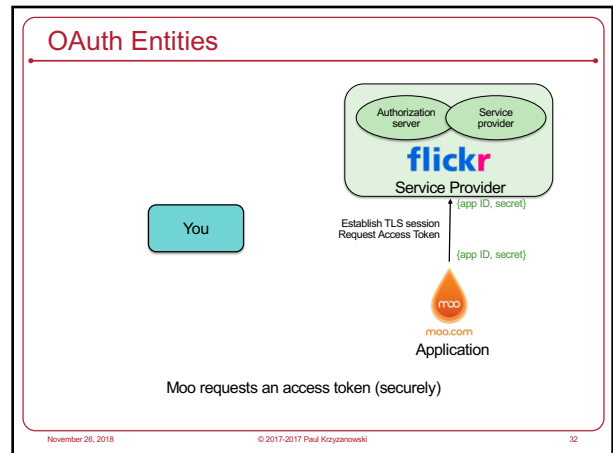
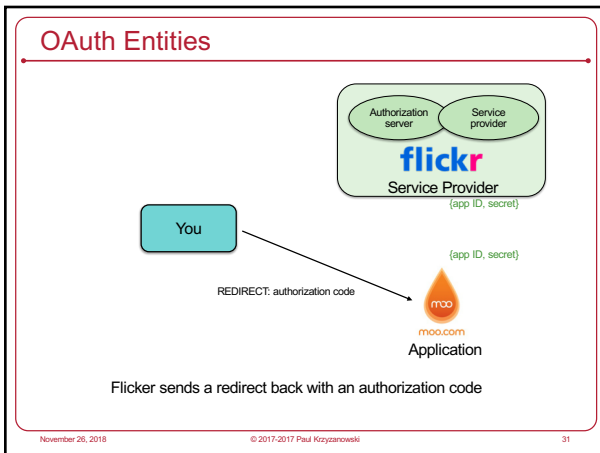
Moo.com app redirects you to the service provider

November 26, 2018 © 2017-2017 Paul Krzyzanowski 29

OAuth Entities

You authenticate (optional) & authorize the request at flickr

November 26, 2018 © 2017-2017 Paul Krzyzanowski 30



Key Points

The slide contains two screenshots. The top one is a Google account login screen with fields for email and password, and a 'Sign in' button. The bottom one is a Google OAuth 2.0 Playground permission screen showing a list of permissions being requested by a third-party service.

- 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/): <https://developers.google.com/oauthplayground/>

November 26, 2018 © 2017-2017 Paul Krzyzanowski 35

Identity Federation: OpenID Connect

November 26, 2018 © 2017-2017 Paul Krzyzanowski 36

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
 - OpenID does not specify authentication protocol – up to provider
 - 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)



November 26, 2018

© 2017-2017 Paul Krzyzanowski

37

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 server (identity provider)
 - Server requests user ID and handles authentication
 - Get back an access token
 - If authentication is successful, the token contains:
 - user ID
 - approved scopes
 - expiration
 - etc.
- } same as with OAuth requests for authorization

November 26, 2018

© 2017-2017 Paul Krzyzanowski

38

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators
 - Used for nonces and session keys

November 26, 2018

© 2017-2017 Paul Krzyzanowski

39

Examples

- Key exchange
 - Public key cryptography
- Key exchange + secure communication
 - Random # + Public key + symmetric cryptography
- Authentication
 - Nonce (random #) + encryption
- Message authentication codes
 - Hashes
- Digital signature
 - Hash + encryption with private key

November 26, 2018

© 2017-2017 Paul Krzyzanowski

40

The End

November 26, 2018

© 2018 Paul Krzyzanowski

41