Overview

- Authentication vs authorization.
- Basic authentication.
- Character encoding.
- Base64 encoding.
- Hashing, encryption.
- Passwords.
- OAuth2.

Authentication vs Authorization

- **Authentication**: Who we are. Example: user-id and password.
- **Authorization**: What we can do. Example: Unix file system permissions.
- **2-Factor Authentication**: Authentication based on 2 independent factors: for example, *what we know* and *what we have*. Examples: user-id password and fingerprint; user-id password and cellphone; user-id password and iris scan.
- Authentication and authorization are orthogonal concepts, but authorization usually requires some kind of earlier authentication.
- **Access control**: control access to resources based on authentication and authorization.

Basic Authentication

1. Client sends a request for some resource which requires authentication.
2. Server replies with a 403 `UNAUTHORIZED` status and a `WWW-Authenticate` header with value `Basic`.
3. Client retrieves user-id and password from its credential cache; if not found in its credential cache, it will get them from the user.
4. Client resends original request, but with an additional `Authorization` header which contains `Basic` along with a base-64 encoding of $UserId : Password$; i.e a base64 encoding of the concatenation of the provided *UserId*, a single colon character :, and the provided *Password*.

Basic Authentication Properties

- Since HTTP is a stateless protocol, `Authorization` header must be sent with each request.
- Client (browser) will cache the authorization for a particular realm so that it can send that authorization whenever it receives a 403 `UNAUTHORIZED` challenge without needing to reprompt the user for user-id and password.
- Base64 is an encoding, not encryption. Hence the password can be read by any intermediary. So basic authentication by itself is not secure and should only be used over HTTPS which provides encryption.

WWW-Authenticate Header

`WWW-Authenticate` *type* `realm=`*Realm*

type	Specifies authentication scheme. Posibilities include `Basic`, `Digest` (extension of basic with nonce and MD5 hashing), `Bearer` (OAuth 2.0 token).
realm	An opaque string specifying the resources being protected. Examples could include `"production server"`, `"CGI scripts"`.

Base64 Encoding

Base64 is used for encoding arbitrary binary data.

1. Partition binary sequence into sequence of 6-bit blocks starting with most-significant bits. If # of bytes not divisible by 3, pad with 0-bytes on right to make a multiple of 3.
2. Map each block into a ASCII alphanumeric (upper-alpha, lower-alpha, digits for a total of 62 possibilities) plus + or / characters.
3. Replace any padding 0's with = characters. Hence a single trailing = indicates 1-byte of padding, 2-trailing = characters indicate 2 bytes of padding.

Hashing

- Map variable length content to fixed-length hash string.
- Good hashing algorithms ensure that changing the content slightly (even by a single bit) will result in a different hash.
- The uses of hashing include summarizing content, cryptographic signatures.
- Example hashing algorithms include MD5 (not cryptographically secure), SHA-256, bcrypt (purposely slow).

```
$ echo hello | md5sum
b1946ac92492d2347c6235b4d2611184  -
$ echo hellp | md5sum
7bd75e0741818d4e6020b0250e52dd46
$
```

Encryption

- Unlike hashing, encryption is reversible.
- **Encryption** converts *plain text* into *cypher text* based on some algorithm and *key*.
- **Decryption** converts cipher text into plain text based on some algorithm and key.
- **Symmetric encryption**: uses the same **secret** key for encryption and decryption. Example algorithms include Blowfish, RC4, AES.
- **Asymmetric encryption**: uses different keys for encryption and decryption.
- **Public-Key Encryption**: Has a public and private key pair. Either key can be used for encryption with the other used for decryption. It is supposedly impractical to derive the private key from the public key (based on 1-way functions like factorization into primes of large numbers). Example algorithms include Diffie-Hellman, RSA, ECC.

HTTPS

- HTTP over Transport Layer Security (TLS).
- Provides authentication of remote web site based on certificate signed by a trusted certificate authority. Certificate provides proof of ownership over a public key.
- Uses public key cryptography to securely setup a session key which is used to symmetrically encrypt the rest of the session. Specifically, client creates a random session key which is then sent securely to the server using the server's public key. Once server decrypts the session key bot client and server share the same session key. All subsequent communition are encrypted using a symmetric encryption algorithm using the session key.

Passwords

- Passwords should never be stored in plaintext.
- Typically, passwords are hashed (not encrypted) using a one-way hashing algorithm like MD5 or SHA256. Only hash is stored. When user logs in, entered password is hashed and compared with stored hash; if they match, then login is allowed, else denied.
- A responsible web programmer should **never** store a plaintext password.
- Initially, Unix passwords were stored in world-readable `/etc/passwd`. Now stored in `/etc/shadow` with restricted readability.
- Simple hash is amenable to *dictionary attack* where hashes are precomputed for many common passwords; if precomputed hash matches stored hash (got by accessing the password file), then password is dictionary word.

Passwords Continued

- Dictionary attacks can be made harder by adding a random salt to each password before hashing. The salt is stored along with the hash to allow matching an entered password. So a 2 character salt with 64 possibilities for each salt character would increase the number of combinations for a dictionary attack by 4096.
- In 2000's, brute-forcing passwords becoming possible because of extremely fast hardware (sometimes using GPUs).
- Normal hashing algorithms like SHA-256 were designed to be fast.
- Current best practice for password hashing is to use purposely slowed up hash algorithms like `bcrypt` to make it harder for crackers.

OAuth2

OAuth2 is a authorization protocol (often misused for authentication).

- A **server** (like twitter or facebook) contains resources (contact details, friends list) owned by a **user**.
- The user authorizes a **client** (an app or third-party web site) to access these resources on her behalf. Hence an app can send out a tweet on behalf of a user.
- OAuth2 is used to set up this authorization.

OAuth2 Protocol Overview

1. Client sends user a request to authorize access to server resource.
2. User interacts with server to authorize request.
3. User returns to client with an *authorization token*.
4. Client sends authorization token to server.
5. Server returns an *access token* to client.
6. Client uses access token to access user resources on server.

OAuth2 Prerequisites

- Client must preregister with server. When registering, it must provide a set of *callback URLs*.
- After registration, server provides client with a `CLIENT_ID` (which is public) and a `CLIENT_SECRET` which is private.

OAuth2 Authorization Request

Client asks user to click on a link which looks something like:

```
https://api.server.com/authorize?
  response_type=code
  &client_id=CLIENT_ID
  &redirect=CALLBACK_URL
  &scope=RESOURCES
  &state=STATE
```

- URL can be anything, but query parameter names are fixed.
- `CLIENT_ID` and `CALLBACK_URL` were set up during the registration process.
- `RESOURCES` specifies the resources on the server the client would like to access on behalf of the user.
- The `state` parameter is optional. If provided, `STATE` is a string whose meaning is interpreted only by the client.

Authorization Grant

If user successfully grants the requested access to the client on the server, then the server **redirects** the user back to the client:

```
CALLBACK_URL?code=AUTH_CODE&state=STATE
```

- `CALLBACK_URL` is the CALLBACK_URL parameter provided during the authorization request.
- `AUTH_CODE` is the authorization code.
- `STATE` is the state parameter provided in the authorization request (if any).
  - The client can include a nonce in the `STATE` parameter to ensure that the callback originated from it.
  - The client can use the `STATE` parameter to redirect the user to different URLs based on the initial request by the user.

Exchanging Authorization Token for an Access Token

The client makes a request to the server:

```
POST https://api.server.com/access_token?
  grant_type=authorization_code
  &code=AUTH_CODE
  &redirect_url=CALLBACK_URL
  &client_id=CLIENT_ID
  &client_secret=CLIENT_SECRET
```

- URL can be anything, but query parameter names are fixed.
- AUTH_CODE is the authorization code granted by the server.
- CALLBACK_URL is exactly the same as what was used when getting the authorization token.
- CLIENT_ID and CLIENT_SECRET are the values set up for the client during the registration process.
- CLIENT_SECRET is omitted if it is a mobile app or single-page browser app since using it would necessarily involve make it public.

Receiving an Access Token

If the exchange is successful, the server responds with a JSON object:

```
{ "access_token": ACCESS_TOKEN,
  "expires_in": EXPIRES_SECONDS
}
```

- `EXPIRES_SECONDS` is the expiry time for the token in seconds.
- The `ACCESS_TOKEN` can be used by the client to make subsequent requests to the server:

```
    Authorization: Bearer ACCESS_TOKEN
```