# Security Mechanisms

For the Web

# Types of security checks

- Obscurity (generally very bad idea):
  - application listens on non-standard port known only to specific people

# Types of security checks

- Obscurity (generally very bad idea):
  - application listens on non-standard port known only to specific people
- Address:
  - where are you coming from? host based access/deny controls

# Types of security checks

- Obscurity (generally very bad idea):
  - application listens on non-standard port known only to specific people
- Address:
  - where are you coming from? host based access/deny controls
- Login:
  - username/password provided to each person needing access

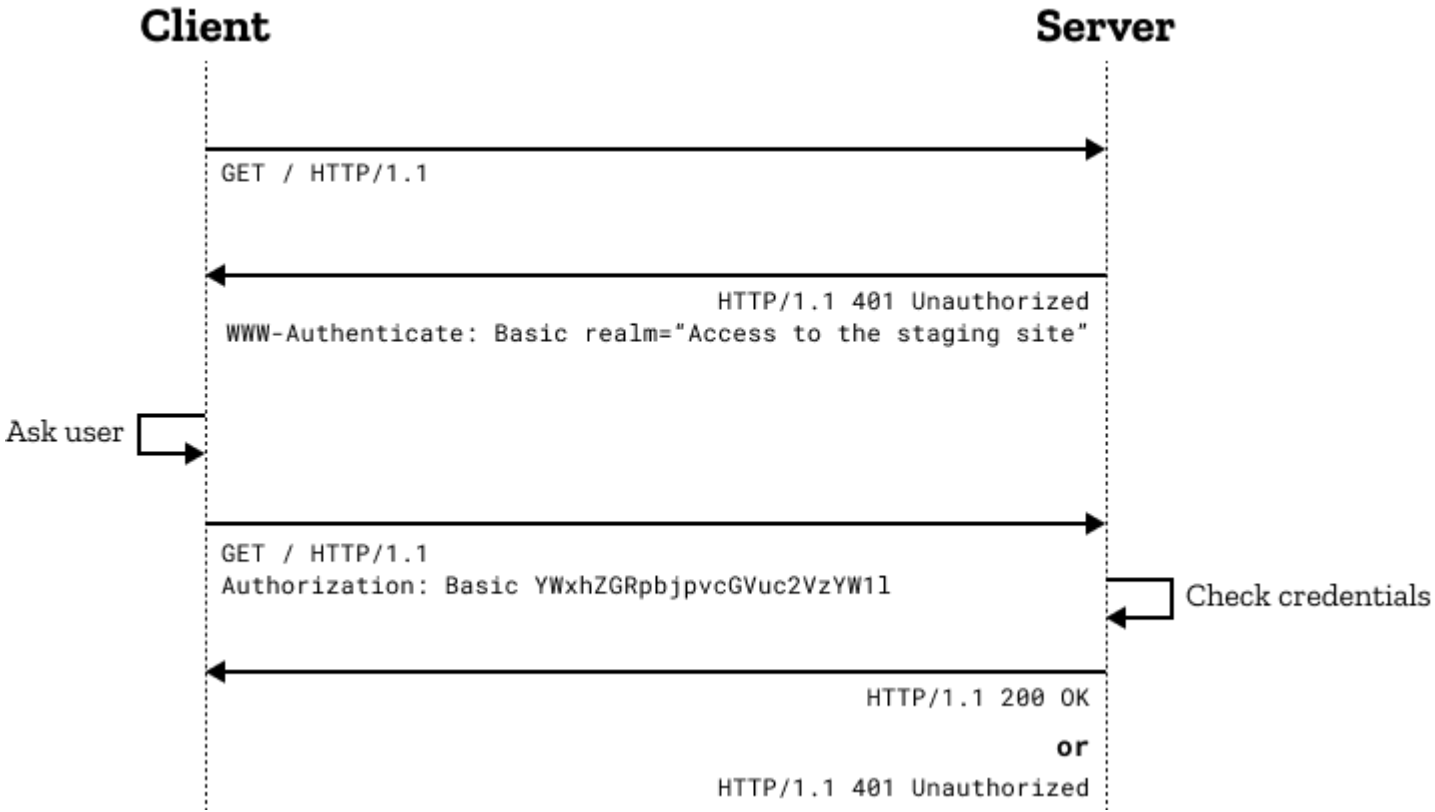
# Types of security checks

- Obscurity (generally very bad idea):
  - application listens on non-standard port known only to specific people
- Address:
  - where are you coming from? host based access/deny controls
- Login:
  - username/password provided to each person needing access
- Tokens:
  - access tokens that are difficult/impossible to duplicate
  - can be used for machine-to-machine authentication without passwords

# HTTP authentication

Basic HTTP auth:

- Enforced by server
- Server returns “401/Unauthorized” code to client
- Contrast with:
  - “404” - not found
  - “403” - forbidden (no option to authenticate)
- Client must respond with access token as an extra “Header” in next request




Src: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

# Problems with HTTP Basic Auth

- Username, Password effectively sent as plain text (base64 encoding)
  - Some minimal security if HTTPS is used (wiretap is difficult)
- Password will be seen in cleartext at server
  - Should not be needed - better mechanisms possible
- No standard process for “logout”

# Digest authentication

- Message digest: cryptographic function
  - eg. MD5, SHA1, SHA256 etc.
- One-way function:
  - f(A) = B
  - Easy to compute B given A
  - Very difficult (near impossible) to compute A given B
- Can define such one-way functions on strings
  - String -> binary number

# HTTP Digest authentication

- Server provides a “nonce” to prevent spoofing
- Client must create a secret value including nonce
- Example:
  - HA1 = MD5(username:realm:password)
  - HA2 = MD5(method:URI)
  - response = MD5(HA1:nonce:HA2)
- Server and client know all parameters above, so both will compute same
- Any third party snooping will see only final response
  - cannot extract original values (username, password, nonce etc)
  - nonce only used once to prevent replay

# Client certificates

- Cryptographically secure certificates provided to each client
- Client does handshake with server to exchange information, prove knowledge
- Keep cert secure on client end
  - Impossible to reverse and find the key

# Form input

- Username, Password entered into form
- Transmitted over link to server
  - link must be kept secure (HTTPS)
- GET requests:
  - URL encoded data: very insecure, open to spoofing
- POST requests:
  - form multipart data: slightly more secure
  - still needs secure link to avoid data leakage

# Request level security

- One TCP connection
  - One security check may be sufficient
  - other network level issues to consider for TCP security
- Without connection KeepAlive:
  - each request needs new TCP connection
  - each request needs new authentication

# Cookies

- Server checks some client credentials, then “sets a cookie”
- Header
  - `Set-Cookie: <cookie-name>=<cookie-value>; Domain=<domain-value>; Secure; HttpOnly`
- Client must send back the cookie with each request
- Server maintains “sessions” for clients
  - Remember cookies
  - Can set timeouts
  - Delete cookie record to “logout”
- Client
  - must send cookie with each request

# API security

- Cookies etc. require interactive use (browser)
- Basic auth pop-up window

APIs:

- Typically accessed by machine clients or other applications
- Command-line etc. possible
- Use “token” or “API key” for access
    - Subject to same restrictions: HTTPS, not part of URL etc.