
Authentication in HTTP using JWT
Information Security – Lecture 10
Aadil Zia Khan

Web and HTTP

- Web page consists of objects
- Object can be HTML file, JPEG image, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL, (Uniform Resource Locator) e.g.,

`www.someschool.edu/someDept/pic.gif`

host name

path name

HTTP overview

- HTTP: hypertext transfer protocol
- Web's application layer protocol
- Follows a client/server model
  - Client: browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - Server: Web server sends (using HTTP protocol) objects in response to requests

HTTP overview (continued)

1. Client initiates TCP connection (creates socket) to server
    1. Server port is always 80 (http) or 443 (https)
    2. Client port will always vary based on the available ports between 49152 and 65535
2. Server accepts TCP connection from client
3. HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
    1. First request is for the base page – all other requests correspond to different objects (e.g., images) in that base page
4. TCP connection closed

- HTTP is "stateless"
    - Each HTTP request is considered an independent request and no information from the previous request is saved

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Problem with Stateless Protocols

- Authentication
  - Suppose you enter your username and password on the login screen
  - The server will check your credentials and send back the homepage
  - Problem: if you navigate to a different page (on the same website), you will have to resend your username and password because the server is not maintaining state
- History of past interaction
  - Suppose you visit an ecommerce site
  - You add items to the shopping cart and navigate to the payments page
  - The server would have no idea of your shopping cart because it is stateless

User-server state: cookies

- HTTP is "stateless" – what if we need to maintain state?

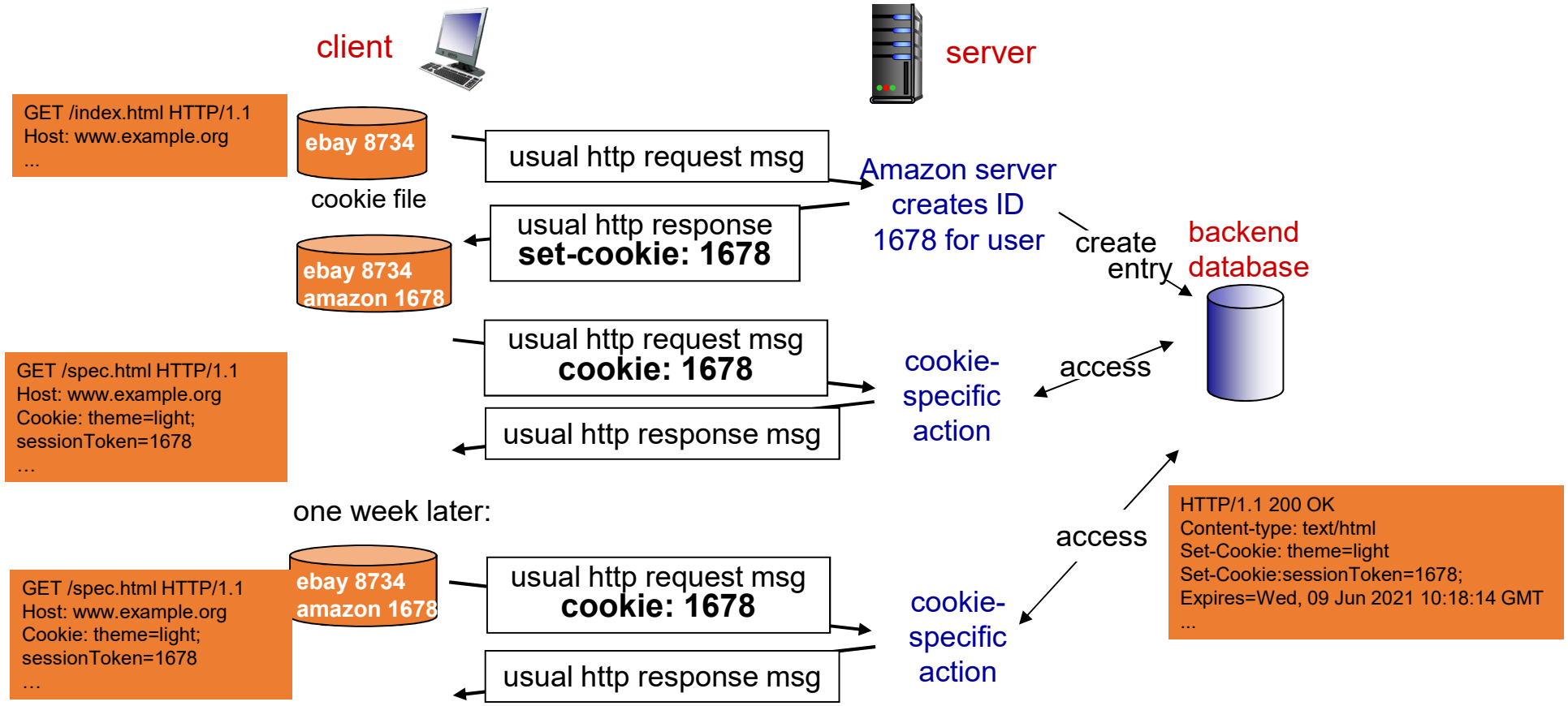
- HTTP uses cookies to maintain state
- Example
  - Authorization
  - Shopping carts
  - Recommendations
  - User session state (Web e-mail)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
  - you may supply name and e-mail to sites
  - you may browse material/products you would want to keep private

Cookies: keeping "state" (cont.)



Session-based Authentication with Cookies

1. The user (browser) sends the login credentials of the user and the info it is requesting
2. The web server authenticates the user and stores all the information about the user in memory/database and returns a sessionId to the user
3. User stores the sessionId in browser cookies - next user request includes the cookie in the HTTP header
4. Web server looks up the sessionId to see if it corresponds to the user
5. If the sessionId is valid and hasn't expired, the web server treats the user as logged in

Session-based Authentication - Limitations

- Performance
  - On each request, the server needs to lookup the sessionId and validate it from the database or memory – this slows down the response times
- Cookie Fraud
  - Eve could gain access to your cookies and then impersonate you on the website – known as a Cross-site Request Forgery attack
- Load Balancing
  - It is possible that your request is split between multiple servers for load balancing purposes – session information (including all transaction details) would need to be saved and synced on each server

Solution: Token-based Authentication

Instead of saving the session related information at the server, encrypt and store it inside a token which will be stored at the client end – not possible in cookies due to size limits and cookie structure

1. Client sends a request to the server with login credentials
2. Server validates the credentials and generates a secure, signed token for the client which includes the required session related information
3. The token is sent back to the client and the browser stores it
4. The client sends the token through the HTTP header each time it sends a request to the server
5. The server decodes and verifies the attached token - if it is valid, the server sends a response
6. The token is destroyed when the session finishes

Token-based Authentication: Benefits

- Scalability
  - Servers do not store any client information - the client sends the complete information through a token inside the web request
- Security
  - Tokens may be encrypted and they expire after some time
- Authorization
  - The token can be created by a third party and include various access rules - then used by the client and server
  - E.g., using Google credentials on a third party website – and specifying that only the contact list may be accessed by the third party

JWT - JSON Web Token

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page.

- It is a web standard
- Defines a way of transmitting information between a client and a server in the form of a JSON object
- A JWT can either be signed (JWS) or encrypted (JWE) or both – else it is called insecure JWT
- A JSON Web Token is basically three strings separated by a dot
    - HEADER . PAYLOAD . SIGNATURE

JWT Header

```
{
        "alg": "HS256",
        "typ": "JWT"
}
```

- This forms the first part of JWT
- The header allows multiple different attributes:
  - alg: the algorithm used to sign or encrypt the JWT
  - typ: the type of token that is being signed or encrypted
  - cty: the type of content
  - kid: a hint indicating which key was used to generate the token signature

- In the above example, HS256 indicates that this token is signed using HMAC-SHA256

JWT Payload

```
{
  "loggedInAs": "admin",
  "iat": 1422779638
}
```

- This is the second part of JWT
- It contains the information needed by the server to identify the user and determine access rights
  - Consists of Claims - pieces of information about an entity
- There are three types of claim fields
  - Registered Claim Names – reserved names
  - Public Claim Names – can be defined at will by the users as long as someone else hasn't already used that name
  - Private Claim Names –producers and consumers of a JWT may agree to any Private claim name that is not Reserved or a Public

JWT Payload - Registered Claim Names

- iss: identifies the entity that issued the JWT
- sub: identifies the entity that is the subject of the JWT
- aud: identifies the recipients that the JWT is intended for
- exp: identifies the expiration time
- nbf: identifies the time after which the JWT become valid
- iat: identifies the time at which the JWT was issued

JWT Signature

- The third part of JWT is the signature
- It is created by combining the header and payload parts of JWT and then hashing them using a secret key

JWT-based Authentication

1. Client sends a request to the server with user login details
2. If the credentials are valid, the server will generate a signed JWT and send it back to the client
    1. Symmetric Signature: a single secret key is used to generate and validate the token – used when there is only one server that signs and validates the token
    2. Asymmetric Signature: server signs using its private key, and shares it with the client - client can now send this token to any application and they can validate it using the public key
3. Client stores the received JWT in the browser
4. Every subsequent request to the server would include JWT
5. Server validates the token, and if it is valid, grants access to the client

Invalidating a JWT

- A JWT may get compromised
- There are two possible actions
  - Wait: JWT has an expiry time – a compromised JWT would be of no use to the malicious user after that
  - Cancel the JWT: to cancel before expiry, client can log off (it would remove the JWT from browser storage) and server can then add the JWT to a blacklist

Hacking the JWT - Modifying the Signing Algorithm

Some libraries used for working with JWT contained logical errors — when receiving a token signed with a symmetric algorithm (e.g., HS256) a public key will be used for verifying the signature

1. When the server creates a JWT, it generates a hash from the header and payload and signs it using the key to create the signature part of the token
    1. HS256 uses a secret key to sign and validate the token **BUT** RS256 uses a private key to sign the token and public key to validate it
2. Attacker will first get hold of someone's token
3. Attacker will change the algorithm in alg parameter from RS256 to HS256
4. Attacker will make changes in the payload as required
5. Attacker will sign the token using the server's public key and send it to the server
6. Since the algorithm is HS256, the server will use the public key as a secret key
    1. Signature would match and the malicious user will get access to the application

Hacking the JWT - Using None

Some libraries allow the signing algorithm to have a value of None

1. The alg claim, specifies the algorithm that is used to sign or encrypt the token
2. If a token has None value in alg claim, then it means that this token need not be validated
3. Any attacker can create a token with alg claim as None and access the resources

slidesgo