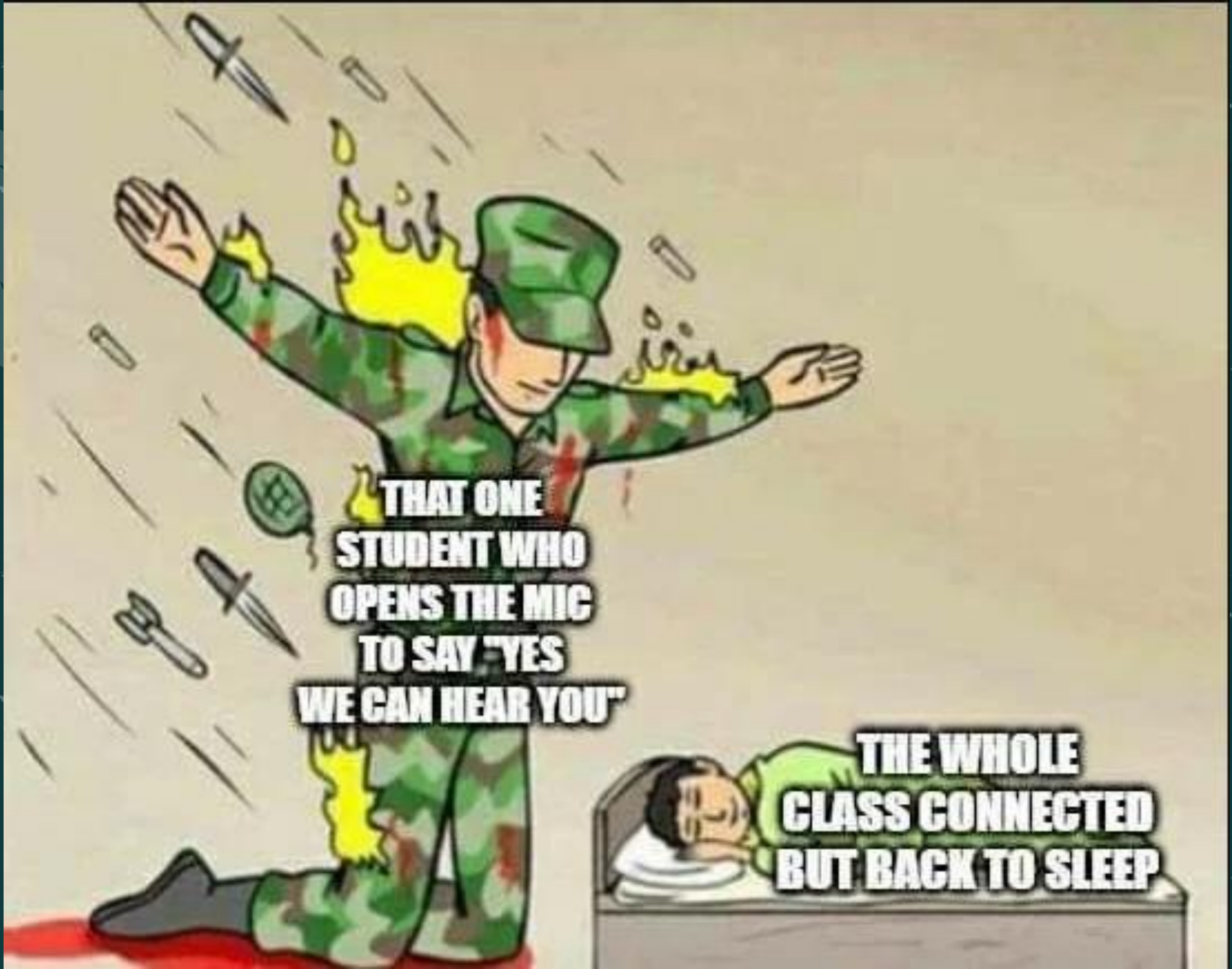
Web Application Security - Cookies

Information Security – Lecture 16

Aadil Zia Khan


THAT ONE STUDENT WHO OPENS THE MIC TO SAY "YES WE CAN HEAR YOU"
THE WHOLE CLASS CONNECTED BUT BACK TO SLEEP

Announcement

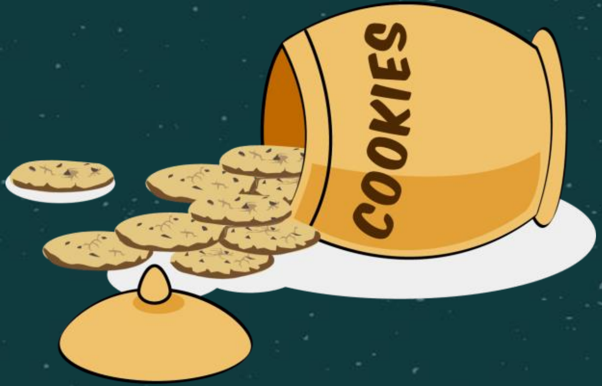
- Midterm for all three sections on Friday – sufficient time would be given
- No lecture on Wednesday / Friday
- It will cover everything up till today's lecture (open books/slides/google)
- Make sure everybody has enrolled on the course LMS page

Cookies - Recap

- HTTP is a stateless protocol
- HTTP cookies are used to maintain state between the browser and the server
  - E.g., maintaining session ID or shopping cart contents across different pages
- Server sends a cookie - contains small bits of data
- Browser stores the cookie and sends it along with future messages to the same server

Do We Need To Secure Cookies

- Cookies may contain extremely sensitive data
  - Session ID
  - Access Token
- If a cookie is captured, the attacker can use it to
  - Impersonate users
  - Gain privileges



Cookies - Recap

Browser sends a first-time request to the server

```
GET / HTTP/1.1
Host: server.com
```

Server reply may include one or more cookies

```
HTTP/1.1 200 Ok
Set-Cookie: access_token=56798123
Set-Cookie: user_id=14
```

Browser’s next request to the server includes cookies

```
GET / HTTP/1.1
Host: server.com
Cookie: access_token=56798123; user_id=14
```

Cookies - Recap

- Additional directives included in the cookies that a server may specify (used to prevent cookie misuse):
  - Expires
    - Specifies the time at which the cookie would become invalid
    - e.g., Set-Cookie: access_token=56798123; Expires=Mon, 26 Apr 2021 03:49:00 GMT)
  - Max-Age
    - Specifies the number of seconds after which the cookie would become invalid
    - e.g., Set-Cookie: access_token=56798123; Max-Age=3600
  - Domain
    - Specifies the domains to which a cookie can be sent
    - e.g., Set-Cookie: access_token=56798123; Domain=trusted.server.com
      - The browser would send the cookie to trusted.server.com but not server.com, nor lms.server.com etc.
  - Path
    - Specifies the exact page (URL) to which a cookie can be sent – stricter trust
    - e.g., Set-Cookie: access_token=56798123; Path=/trusted/path

Trackers and Supercookies

- Suppose a server includes the following cookie in response to a browser’s HTTP request
  - Set-Cookie: access_token=56798123; Domain= .com
  - Cookie is set on the top-level domain (TLD) instead of a specific domain
  - This is called a supercookie or a tracker

- Do you see any problem in this example?
  - Anytime the browser sends an HTTP request to any domain that ends in .com the cookie would be sent along with it
  - Privacy: every website belonging to the .com domain would be able to track the user
  - Information leakage: a sensitive piece of data stored inside that cookie would be available to all other sites in the .com domain

Trackers and Supercookies – Etag Tracking

- Browsers block supercookies because of user's privacy/security concerns
  - Trackers have found a way to beat that – using Etags
  - Etags were actually created to enable caching

- ETag or Entity Tag (inside HTTP headers) can be used to identify browsers accessing a resource
  - It is a unique id assigned to a resource (on the webpage) by the server
- ETags are cached by the browser, and returned to the web server when the said resource is requested a second time
- This enables websites to track users across sessions, in spite of changing the IP address, disabling JavaScript, cookies and/or local storage
  - This is achieved as a result of sending the Etag data in the http-header – e.g. when an advertisement banner (belonging to the same advertising broker) is loaded on different pages

Protection Against Trackers

- Browsers block supercookies
- In case of Etags, clearing the browser cache does the trick

Next Step

- Now Lets Look At How We Can Secure Cookies – when we are not getting rid of them altogether

Preventing Man In The Middle Attacks On Cookies

- If a cookie is exchanged via HTTP, it is in plaintext and vulnerable to MITM attacks
    - By capturing the cookie, the MITM can hijack the session

- How can we protect the cookie when the server is sending it
    - Use HTTPS – the HTTP response packet (including the header) would be encrypted

- How can we ensure that the browser never sends the cookie unencrypted
    - The server adds a “Secure” flag to the cookie – this tells the browser to never send the cookie in plain HTTP requests

Preventing Cookie Leak In An XSS Attack

- A malicious Javascript (injected through a Cross Site Scripting attack) can read the cookie stored locally using the document.cookie property

- The server adds a “HttpOnly” flag to the cookie
  - This instructs the browser not to allow Javascript to access the cookie

Preventing Cross-Site Request Forgery

- What is CSRF?
- Web browsers automatically and invisibly include any cookies used by a given domain in any web request sent to that domain
- Problem?
  - Suppose a user’s browser has a cookie which automatically signs him into nationalbank.pk
  - Suppose the user visits a website fancyanimation.com which tricks him into clicking (clickjacking) on a link which sends a transaction request to the nationalbank.pk server
  - The browser will send the cookie with the request – the server will execute the transaction request since the cookie is authentic

Preventing Cross-Site Request Forgery

- To prevent CSRF
- The server (nationalbank.pk) adds a “SameSite” flag to its cookie
- The server (nationalbank.pk) tells the browser not to include the cookie in requests that were generated by different origins
    - When the browser initiates a request to nationalbank.pk and its cookie is tagged as SameSite, the browser will first check whether the origin of the request is the same origin – else the browser will not include the cookie in the request