7️⃣

Chapter 7 - Attacking session management

Attacking Session Management

The Need for state

- HTTP is stateless
- unique session tokens are used to link requests to a user
- The two main vulnerabilities to do with states:
    - Weakness in the generation of session tokens
    - Weakness in the handling of session tokens

Alternatives to sessions

- HTTP Authentication
- the browser will resubmit the credentials with each request
- Sessionless state mechanisms
- The app sends all the necessary information at once in a “binary blob”

Weakness in Token Generation

- Weak tokens are not just for login, they can apply to:
    - Passwords recovery tokens
    - Tokens placed in hidden form fields to prevent csrf
    - tokens used to give one-time access to protected resources

    - persistent tokens used in “remember me” functions
    - tokens allowing customers of a shopping application that does not require accounts to retrieve the current status of an existing order

Meaningful tokens

- Some tokens are just parts of the user’s info encoded. This can be used to retrieve ids, usernames, roles etc

Predictable Tokens

- Sequential tokens are very bad
- Concealed sequences are bad
- Time dependant tokens are bad
- Weak random number generation are bad

Concealed sequences

- Say a site uses the seemingly random tokens `lwjVJA`, `Ls3Ajg`, `xpKr+A` and `XleXYg`
    - If these are base64 decoded it is binary gibberish
        - If this binary is conerted to hex we get `9708D524`, `2ECDC08E`, `C692ABF8`, `5E57962`, this also looks random
        - If we subtract each number from the previous one we get `FF97C4EB6A`, `97C4EB6A`, `FF97C4EB6A`, `97C4EB6A`. A repeating pattern
- From this we can tell that the algorithm they use, generates tokens by adding 0×97C4EB6A to the prior value, truncates the result to 32-bit number, Base64 encodes it, and then uses it
- This means we now know all prior and future tokens

Time Dependancy

- if the token is based on the time of creation and not enough entropy has been applied, this can be detected by sending multiple creation requests quickly
- this can lead to easy bruteforcing

Weak Random Number generation

- True random is impossible to generate on a computer

- knowing how the “random generator” works can lead to breaking it.
- Vulns have been found in common frameworks before

Testing the quality of randomness

- You may need to apply hypothesis testing in order to tell if a token is random or not
- this can require at minimum 100 tokens, at max 20,000.

Encrypted Tokens

- This can be decrypted/cracked dependin on the algorithm

ECB Ciphers

- “Electronic Cookbook Cipher” = ECB
- symetric encryption
- plain text may still be visible in the cipher
- The same word will be encryped in the same random text. Repeating random text indicates repeated plaintext

CBC Ciphers

- “Cipher block chaining” = CBC
- The plain text is XORed with an Initialisation vector, which forms a block, this block is used as the IV for the next part of the plaintext, repeat.

Weaknesses in Session Token handling

Disclosure of Tokens on the Network

- This is a whole section on MitMs which is irrelivent since this is always out of scope

Disclosure of Tokens in logs

- LFI can lead to tokens disclosure

Vulnerable Mapping of Tokens to sessions

- There is no reason why a user would be simultaneousl using to session tokens

- Static tokens, ones that are one per user and dont terminate, are bad.
- tokens may be valid for any user as long as they are valid for one. (bad)

Vulnerable Session Termination

- reduces the time an old vulnerable token can be used
- allows for an existing session to be terminated when it is no longer required
- ways that logout functionality is not effective:
    - Not implemented at all
    - does not cause the server to invalidate the cookie, just removes the user’s cookie
    - not communicated to the server on logout, simply executes a clientside script to remove cookie

Client Exposure to Token Hijacking

- xss to steal session tokens
- make a session token (so you know it) and then make another user login using it
- csrf exploits the browser automatically giving the token

Liberal Cookie Scope

- The cookie mechanism can aos be used to specificy the domain and url path to which each cookie will be resubmitted, this uses the `domain` and `path` attributes included in the `Set-cookie` instruction

Cookie Domain Restrictions

- The browser will only submit cookies to the same domain, and its subdomains, eg `Set-cookie: id=123; domain=example.com` will be a valid cookie on example.com and all of its sub domains
- a token specifying one subdomain is valid on another subdomain

Cookie Path Restrictions

- an app that has set a cookie with a specific path will be valid for subdirectories, but not for parent directories

Securing Session Management

Generate Strong Tokens

- use an extremely large set of possible values
- strong sources of pseduorandomness
- should not be bruteforcable
- should not be gusseable
- Examples of pseduorandom items that can be **ADDED** to the already complex token to add entropy:
    - Source IP and port
    - User agent
    - Time of request in miliseconds

Protect tokens throughtout their life cycle

- Only ever transmitted over HTTPS
- never transmitted in url
- logout functionality should dispose of all sessions
- Session expiration should be implemented
- Concurrent logins should be prevented, a different session token should be used each login.
- domain and path scope should be as tight as possible
- the apps codebase should be regourously audited
- any unknown tokens should be immedietly rejected
- csrf can be prevented with 2fa or recauth before critical actions