Attacking Session Management

- Session Management is a fundamental mechanism used in web apps
- Any app that employs authentication uses session management to uniquely identify and track a user's interaction with the app

- The Need for State

→ HTTP itself is a stateless protocol, subsequent messages are treated are mere transactions.
→ But sessions are implemented through either through third party API's or frameworks. Some developers choose to use a bespoke solution.
→ Most Common solution to session management is the Cookie Method. The first time a user visits a web app, the
⇒ Set-Cookie header sends a token or identifier, this is resubmitted is following client requests
⇒ Ex: Server == Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55
◇ Client == Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55
→ Vulnerabilites in session maanagement mech fall into
⇒ Weakness in generation of session tokens
⇒ Weaknessses in the handling of session tokens throughout their life cycle.

- Alternatives to Session

→ HTTP authentication
⇒ In HTTP authentication, the user's credentials are resubmitted in a secured format in headers of the subsequent requests.
⇒ Rarely used in complex applications
→ Sessionless State mechanisms
⇒ Some apps do not use session tokens, but rather transmit all state data via the client in a cookie or a hidden form field.
⇒ The data transmitted must be properly protected

- Weaknesses in Token Generation
-
→ Session management mechanisms are often vulnerable to attack because tokens are generated in and unasafe manner

⇒ Meaningful Tokens
• Some session tokens are created using a transformation of user data
• This can lead to reversal of obfuscated string and hence forgery of the token to hijack a user's session
• They often can meaningful data that creates a structure. Some components in a token might be
◇ Username
◇ Account_id
◇ Users's name
◇ User's E-mail
◇ The user's role or group
◇ A timestamp
◇ A sequential number
◇ The client's IP address

⇒ Predictable Tokens
• Even if tokens do not contain meanigful data, sequences and patterns can be predictable
• Using an automated attack these potential tokens can be verified
• They commonly arise from,
◇ Concealed Sequences
▪ It is common to encounter session tokens that contain concealed sequences, to decipher these they need to be de-obfuscated and put into a correct format. Once the logic is discovered you can script an attack to predict future tokens.
◇ Time Dependency
▪ Some apps will employ algorithms to generate sesion token using the current sys time.
▪ If enough tokens are captured and analyzed, there are possibilities to crack this generation method
◇ Weak random number generation
▪ If the random number generation logic is exposed or is easily extrapolatable from the tokens themselves, this creates weak session tokens

⇒ Encrypted Tokens
• Some apps employ an encrypted token but in some situations depending on the encryption algorithm it may be possible to tamper with the token and this depends on the crypto alg used.

◇ ECB Ciphers
- Using an electronic codebook cipher takes the plaintext breaks it into blocks and individually encrypts it.
- While decryption, the ciphertext is again broken into blocks and decrypted individually.
- But since a plaintext always encrypts to certain block, duplication attacks can break the token processing.
- Everything depends on how the application processes the individual token data

◇ CBC Ciphers
- With a CBC cipher, each block of plaintext before being encrypted is XORed against the preceding block of ciphertext.
- This prevents identical plaintext into encrypting into identical cipher blocks
- The XOR is reversed in decryption phase
- Modifiying the recieved token bit by bit and observing the behaviour after resubmiting the token can lead to a privilege escalation
- Try bitflipping

- Weaknesses in Session Token Handling

→ Disclosure of Tokens on the Network

⇒ A positioned attacker can sniff plaintext cookies or cookies transmitted through http

→ Disclosure of Tokens in Logs

⇒ Plaintext exposure of session tokens in logs can present an oppurtunity for them to be compromised

⇒ Ex: jsessionid in headers

→ Vulnerable mapping of session tokens

⇒ Avoid multiple sessions

⇒ Avoid Static tokens

→ Vulnerable Session Termination

⇒ Termination is important because,
- Keeping life span of session helps shorten the windows of oppurtunity for a session hijack to be efficacious.
- Also provides an user a means to invalidate sessions that are no longer needed

⇒ Flawed Logout functionality
- No invalidation of session
- Set-Cookie Blank . But server processes requests with old cookies
- Logout is not communicated to the server. But diverted using a client side script.

→ Client Exposure to Token Hijacking

⇒ An attacker can target other users in an attempt to capture sesion token
- Using XSS to query session tokens
- CSRF and session fixation

→ Liberal Cookie Scope

⇒ Cookie Domain Restrictions
- Avoid lliberal cookies as sensitive cookies can be sent over to unnecessary location from where they may be susceptible to attacks

→ Cookie Path Restrictions

⇒ Avoid permitting other directories to access your cookies as these might also expose the cookies to an attack surface

- Securing Session Management

→ Generate Strong Tokens

⇒ The most effective token generation mech(s) are
- Use extremely large set of possible values
- Contain psuedorandomness, ensuring an even and unpredictable spread of tokens
- All data about the session's owner should be on the server in the session object to which the token corresponds

→ Protect Tokens Throughout Their Life Cycle

⇒ Token should be strictly HTTPS only.

⇒ Cookies should be scope controlled and any cookie trasmitted over HTTP should be trated as tainted

⇒ Never use session tokens in the URL, prefer a POST req and store token in a hidden field of an HTML form for apps that have cookies disabled

⇒ Session termination should be possible

⇒ Concurrent logins should be prevented. New session tokens to be issued for new logins and old session token to be discarded similar to a

logout

⇒ Specific measures to defend session management
- Rid the codebase of XSS
- Arbitrary tokens submitted by user are to be rejected and the user should be redirected to login page
- CSRF attacks to be made difficult by enforcing two step confirmation or reauthentication
- Fresh sessions to be created after succesful authentication, to mitigate the effects of session fixation attacks.

→ Per Page Tokens

⇒ Issuing and validating per page tokens and their sequence and the main session token as a whole can prevent session management attacks on a whole

→ Log, Monitor and Alert

⇒ Session management should be closely integrated with its logging and alerting mech(s)
- Monitor invalid tokens.
- Brute-force attempts against the sesion tokens
- Alering users about anamolous events related to their session

→ Reactive Session Termination

⇒ Some apps terminate sessions agresively over small modifications

⇒ Although they are good in a way, they can get in the way of an actual pentest.