Web Assessment

- Injection: untrusted unsanitized data inserted in an interpreter (SQL, OS, XML, JS). Can lead to execution of unintended commands/access data without proper authorization
  - Exploitability: easy
  - Prevalence: common
  - Detectability: easy
    - Static code analysis tools: find uses of interpreters, trace data through appliation. Can be automated in CI builds.
    - Manual code reviews
    - Penetration Testing to validate vulnerabilities by building exploits
    - Dynamic analysis useful to test error conditions
  - Impact: severe
  - Prevention
    - Sanitize untrusted data
    - Use safe APIs (parametrized interfaces)
    - Use interpreter-specific escape routines on all parameters
    - Whitelist all inputs
  - Secure programming principles
    - Architect and design security policies
    - Adopt secure coding standards
    - Least privilege
    - Pay attention to compiler and static analysis tools warnings
    - Sanitize inputs
    - Sanitize data sent to others
    - Use effective quality assurance techniques
- Javascript run inside browser. Most bugs are breaches of
  - Same Origin Policy: prevent malicious scripts to access sensitive data from DOM of other pages
  - Sandbox mechanism: isolate running programs to mitigate system failures, limit spread of vulnerabilities, isolated execution of untrusted code. Controlled set of resources (disk space, memory, limited network and file system access
- Cookies
  - Structure
  - Types
    - First party
    - Third party (cookies from servers not directly visited)
    - Session: deleted when browser quits
    - Permanent: could survive days/months
  - N.B. information contained in cookie must never be sensible because it can be manipulated by the user (untrusted)
  - Session/cookie hijacking: attacker steals information that can be used to authenticate as the victim
- Insecure Deserialization: deserialize tampered object supplied by attacker
  - Exploitability: difficult
  - Prevalence: common
  - Detectability: average
    - Object and data structure related attacks. Tamper application logic/remote code execution. Classes change behavior during/after deserialization
    - Data tampering attacks (access control attacks). Content of the used data structure is changed
  - Impact: severe
  - Attacks
    - Java Serial Killer: RCE on server
    - PHP object serialization of cookie containing sensitive information: gain admin privileges
  - Prevention
    - Safe architectural pattern: don't accept serialized objects but only primitive data types (typically not possible)
    - Implement integrity checks on serialized objects to prevent data tampering
    - Enforce strict type constraints during deserialization, before object creation (bypassable)
    - Isolate deserializing code (low privilege environment)
    - Log deserialization exceptions, failures
    - Restrict/monitor network connectivity from deserializing servers/containers
    - Monitor deserialization activity
  - Secure Programming Principles
- Cross Site Scripting XSS: application creates page/use browser to create page including untrusted unvalidated input. Allows attacker to hijack user session/rce on victim's browser, malware delivery, website defacement
  - exploitability: easy
  - prevalence: widespread
  - detectability: easy
  - Impact: moderate
  - Prevention
    - Sanitize untrusted data
    - Use frameworks that escape XSS by design (React, Ruby on Rails)
    - Escape untrusted http request data
    - Apply context-sensitive encoding when modifying browser document client-side
    - Web Application Firewall with XSS presention scripts (modSecurity)
  - Secure Programming principles