DataArt

Application Security Testing Checklist

Authentication

- [ ] **Password Quality Control**
  Check whether users are required to set sufficiently strong passwords.
- [ ] **Password Brute Force**
  Ensure login, password change, and password recovery functionality are protected from brute-force attacks.
- [ ] **Username Enumeration**
  Confirm whether the app does not disclose the reason for a failed login either though explicit message or indirectly, e.g. via different response timing.
- [ ] **Credentials Distribution**
  Test if distribution channel is secure and users are required to reset/change their password upon first login.
- [ ] **Credentials Storage**
  Check that credentials are stored in a manner that does not allow their original values to be recovered, using modern algorithm with key stretching such as PBKDF or bcrypt.
- [ ] **Change Password**
  Confirm whether a password change function is implemented, requires entry of the current password and notifies the user about the password change event via any out-of-band channel.
- [ ] **Forgot Password**
  Validate the password recovery mechanism does not expose any weaknesses such as email enumeration, reusable or unrestricted recovery URL, use of guessable security questions, etc.
- [ ] **Remember Me**
  Test that the "remember me" functionality memorizes only non-secret items i.e. usernames.
- [ ] **User Impersonation**
  Verify that user impersonation functionality, if exists, is strictly controlled to ensure that it cannot be misused.
- [ ] **Single Sign-On**
  For apps with SSO support, ensure that login workflow follows security recommendations of the corresponding SSO protocol and exchanged tokens are secure.
- [ ] **Multistage Login**
  For app with multistage login process (such as MFA), make sure that app does not contain logic flaws that allow attackers to skip stages, modify previously submitted data on later stages, etc.
- [ ] **Logging and Monitoring**
  Test that all authentication-related events (login, logout, password change etc.) are logged.

Session Management

- [ ] **Token Generation**
  Ensure session tokens have sufficient entropy and they cannot be predicted or extrapolated by an attacker.
- [ ] **Token Transmission**
  Confirm that tokens are transmitted exclusively over HTTPS protocol. When tokens are stored in cookies, make sure Secure, HttpOnly and SameSite flags are set on these cookies.
- [ ] **Token Disclosure**
  Ensure session tokens do not appear in URLs or other places that could potentially be accessed by attacker.
- [ ] **Session Fixation**
  Verify that a fresh session token is issued when a user transitions from anonymity to authenticated.
- [ ] **Session Termination**
  Check whether the logout functionality disposes all session resources and invalidates the session token. Also ensure the sessions are expired upon inactivity with a reasonable timeout.

Access Control

- [ ] **Broken Access Controls**
  Ensure that the application's access controls are sufficient to prevent unauthorized access and privilege escalation, both vertical and horizontal.
- [ ] **Static Resources**
  Ensure unauthorized access to any static files is not possible.
- [ ] **Multistage Functions**
  Test whether all pages of a multistage process employ proper access control checks.
- [ ] **Administrative Access**
  Check whether administrative access is additionally protected with MFA or IP filtering.

Back-End Attacks

- [ ] **SQL Injection**
  Ensure that all database calls use prepared statements (aka parameterized queries).
- [ ] **OS Command Injection**
  Ensure that operating system commands are called out indirectly using corresponding APIs.
- [ ] **Code Injection**
  Verify that user-supplied input (or data derived from this input) is not passed into dynamic execution.
- [ ] **Path Traversal**
  Ensure that user-submitted data is not passed to any file system API directly.
- [ ] **File Upload**
  Check whether an application restricts file upload with a whitelist of allowed file types, a maximum size for files, and it performs virus checks on uploaded files.
- [ ] **Buffer Overflows**
  For apps that use unmanaged runtime, verify for buffer overflow, number overflow and format string vulnerabilities.

Client Attacks

- [ ] **XSS**
  Ensure that the user-supplied input is properly encoded when it is output to a HTML element content or attribute, JavaScript string, URL string or CSS property.
- [ ] **CSRF**
  Test whether the application uses per-page tokens to prevent CSRF attacks.
- [ ] **Same-Origin Policy**
  Ensure that the scope of the policy is as restrictive as possible.
- [ ] **Unvalidated Redirects**
  Verify that application does not use user input as part of redirect URLs.
- [ ] **Host Header Attack**
  Confirm that the host header is not used when generating links or URLs within the application.

Client-Side Controls

- [ ] **Data Validation**
  Check whether client-generated data is validated on the server-side of the application.
- [ ] **Data Transmission**
  Verify that critical data is not transmitted via the client without integrity control.

Information Disclosure

- [ ] **Verbose Error Messages**
  Confirm that verbose error messages or debug information never returns to the user's browser.
- [ ] **Published Information**
  Ensure that information that may be of use to an attacker, including usernames, log entries, etc., is not published.
- [ ] **Client-Side Code**
  Ensure the client-side code does not reveal any useful information such as developer names, revisions, dates, etc.
- [ ] **Server Banners**
  Check whether server banners are removed or modified to minimize the disclosure of specific software version information.

Local Privacy

- [ ] **Persistent Cookies**
  Ensure that sensitive data is not stored in a persistent cookie.
- [ ] **Cached Web Content**
  Verify that suitable cache directives are used to prevent sensitive data from being stored by browsers.
- [ ] **Browsing History**
  Confirm that an application does not use URL strings to transmit sensitive data.
- [ ] **Auto-Complete**
  Test whether autocomplete is disabled within forms that collect sensitive data.
- [ ] **Local Storage**
  Verify that sensitive data that needs to be stored locally is encrypted to prevent easy access.

Application Server

- [ ] **Default Content**
  Confirm that all default content and functionality not required for business purposes is removed.
- [ ] **HTTP Methods**
  Check that all methods – other than those used by the application (typically GET and POST) – are disabled.
- [ ] **HTTP Proxy**
  Ensure that the web server is not configured to run as a proxy.
- [ ] **Transport Security**
  Verify that only high-grade ciphers and protocols are supported for negotiating encrypted sessions over TLS.
- [ ] **Secure Headers**
  Check whether special HTTP response headers are in place to increase security (Strict-Transport Security, X-Frame-Options, X-XSS-Protection, X-Content-Type-Options).