Web Application Security Checklist V1

Category	Name	Description
Configuration and Deploy Management	**HTTP Methods**	Ensure that the web server does not support the ability to manipulate resources from the Internet (e.g., PUT and DELETE).
	HTTP Strict Transport Security	Implement HTTP Strict Transport Security (HSTS) HTTP header in the web server to ensure all communication from a browser is sent over HTTPS (HTTP Secure) and prevent man-in-the-middle attacks.
	HTTP X-XSS-Protection	Implement X-XSS-Protection HTTP header in the web server to prevent XSS (Cross-Site Scripting).
	X-Frame-Options Header	The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a <frame>, <iframe>, <embed>, or <object>. Use the X-Frame-Options header to prevent Clickjacking vulnerability.
	HttpOnly Flag	Mark all of the important cookies as HTTP Only in the web server. Using the HttpOnly flag when generating a cookie helps mitigate the risk of client-side script accessing the protected cookie.
	Known Vulnerabilities / Security Patches	Ensure that known vulnerabilities that vendors have patched are not present.
	Back-up Files	Ensure that no back-up files of source code are accessible on the publicly accessible part of the application.
	Web Server Configuration	Ensure that common configuration issues such as directory listings have been addressed.
	Infrastructure Admin Interfaces	Ensure that administrative interfaces to infrastructure, such as web servers and application servers, are not accessible to the Internet.
	Application Admin Interfaces	Ensure that administrative interfaces to the applications are not accessible to the Internet.
	Data Protection Transport - TLS Version	Ensure to use the latest TLS version.

Source: https://github.com/MohammedAljuhani/Web-Application-Security-Checklist

Web Application Security Checklist V1

Category	Name	Description
Access Control	**Authorization**	Ensure that resources that require authorization perform adequate authorization checks before being sent to a user.
	Authentication	Specify a minimum password length of at least eight characters. Longer passwords provide more protection against brute force attacks. Complex passwords are requiring mixed character sets (alpha, numeric, special, mixed case).
	CAPTCHA	Use a CAPTCHA on any web Forms (e.g., Login page and Registration page) to prevent automated attacks.
	Account Lockouts After Failed Attempts	The most obvious way to block brute-force attacks is to lock out accounts after a defined number of incorrect password attempts. Account lockouts can last a specific duration, such as one hour, or the accounts could remain locked until manually unlocked by an administrator.
	2-Factor Authentication (2FA)	Implement 2-Factor Authentication (2FA) on critical functions such as login and admin page to reinforce authentication/authorization.
	Authorization Parameter Manipulation	Ensure that once a valid user has logged in, it is not possible to change the session ID's parameter to reflect another user account.
	Session Timeout Management	Session timeout management and expiration must be enforced server-side: • Set session timeout to the minimal value possible depending on the context of the application. • Avoid the "infinite" session timeout. • Prefer the declarative definition of the session timeout in order to apply global timeout for all application sessions. • The application has to track the inactivity time server-side and, after the timeout is expired, automatically invalidate the current user's session and delete every data stored on the client.

Source: https://github.com/MohammedAljuhani/Web-Application-Security-Checklist

Web Application Security Checklist V1

Category	Name	Description
	Change Default Account Passwords	Default accounts are often the source of unauthorized access by a malicious user. When possible, they should be changed immediately upon installation and configuration of the system or application.
Data Protection	**Data Storage**	Ensure data is protected to ensure its confidentiality and integrity, where required.
	Passwords	Passwords must be hashed before getting stored in the database.
	Sensitive Data in HTML	Ensure that there is no sensitive data in the HTML
Input Validation	**Input Injection**	Sanitize and filter user inputs to prevent web attacks such as SQL injection and XSS (Cross-Site Scripting).
	Cross-Site Request Forgery (CSRF) Protection	CSRF is an attack vector that tricks a web browser into executing an unwanted action in an application to which a user is logged in. Check if your framework has built-in CSRF protection, and use it. If the framework does not have built-in CSRF protection, add CSRF tokens to all state-changing requests (requests that cause actions on the site) and validate them on the backend.
	Encoding	Encode all input data before sending the response to the browser
Error Handling	**Application Error Messages**	Ensure that the application does not present application error messages to an attacker that could be used in an attack.

Source: https://github.com/MohammedAljuhani/Web-Application-Security-Checklist

References

- https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Web_Application_Penetration_Checklist_v1_1.pdf
- https://www.process.st/checklist/application-security-audit-checklist-template/
- https://github.com/0xRadi/OWASP-Web-Checklist

Source: https://github.com/MohammedAljuhani/Web-Application-Security-Checklist