Web Security Guidelines

Internal vs External

Limit exposing any content (e.g., data, assets, etc.) to the Internet (public network) unless necessary. Intranet (private network) should be used to restrict content to only company employees. Use of a VPN would be required to gain access to Intranet from external location.

Restricting Bots

Use a robots.txt file at the root of the site to restrict the routes that (legitimate) crawlers can search through. This will improve site performance and limit unwanted site content from leaking into search results. It is possible to restrict crawling to the entire site as well through the robots.txt file.

HTTP Security Headers

All sites should configure the web server to output the following HTTP response headers to mitigate attack vectors on the web client.

- Strict-Transport-Security
- Content-Security-Policy
- Permissions-Policy
- Feature-Policy (made obsolete by Permissions-Policy)
- Referrer-Policy
- Server
- X-Frame-Options
- X-Content-Type-Options
- Cross-Origin Resource Sharing (CORS)
    - Access-Control-Allow-Origin
    - Access-Control-Allow-Credentials
    - Access-Control-Allow-Headers
    - Access-Control-Allow-Methods
    - Access-Control-Expose-Headers
    - Access-Control-Max-Age
    - Access-Control-Request-Headers
    - Access-Control-Request-Method
    - Origin
    - Timing-Allow-Origin
- Cross-Origin-Resource-Policy
- Cross-Origin-Embedder-Policy
- Cross-Origin-Opener-Policy
- Expect-CT (obsolete in newer browsers)
- X-XSS-Protection

To further harden your security, consider adding **Public-Key-Pin** headers (HPKP), however this can break your site if you forget to update the header value whenever your TLS certificate changes.

Use https://observatory.mozilla.org and https://securityheaders.com to help evaluate any flaws in the configuration of HTTP security headers.

HTTPS

Every web request should use the HTTPS scheme. This will ensure that the web traffic between the browser and the web server is secure.

Do	Don't
 <script src="https://company.com/cat.js" /> <link href="https://company.com/cat.css" /> Back ONLY using HTTPS will avoid Mix Content Mode in the browser, which leaves the site vulnerable to information leakages.	 <script src="http://company.com/cat.js"></script> <link href="http://company.com/cat.css" /> Back

Internal Web Resources (e.g., Styles, Scripts, Images, etc.)

Each application should be serving internal assets from the same origin because of the following:

- Simplifies the configuration of HTTP Content Security Policy (CSP) headers
- Improve management of assets through consolidation
- Reduce troubleshooting efforts
- Improve performance through less CORS preflight requests

Do	Don't
  Serving all company assets from ***exactly the same*** origin (in purple) will improve overall site performance, security and maintainability.	

Third-Party Web Resources (e.g., Styles, Scripts, Images, etc.)

Avoid loading third-party assets unless required to do so. Instead, save a **vetted local** copy that is served from a trusted internal server.

Do	Don't
<script src="cdn.company.com/js/fancybox.js"></script> <link href="cdn.company.com/css/fancybox.css" /> ONLY use an external CDN for loading assets if the performance from the geo-affinity is required, otherwise, it's a potential supply chain attack vector. Using a **vetted copy** from an internal CDN would limit security exposure. Don't forget to use **HTTPS**!	<script src="cdn.jsdelivr.net/fancybox.js"></script> <link href="fonts.fontawesome.com/fancybox.css" />

If CDNs ***MUST*** be utilized, then you must ensure the CDN:

- Supports Access-Control-Allow-Origin
- All external resources must be decorated with "**integrity**" and "**crossorigin**" attributes
    - integrity: string representing the hash function AND hash digest of the external resource file
    - crossorigin: Set to "anonymous" to restrict cookies from being sent for anonymous requests

```
<script
  src="https://cdn.jsdelivr.net/fancybox.js"
  integrity="sha256-v90qNGx6fS48N0tRxiGkqveZETq72KgDVJCp2TC"
  crossorigin="anonymous"></script>
```

Inline Styles and Inline Scripts

All inline style and scripts are considered unsafe due to exploitable XSS attacks. Instead, load styles from CSS files and scripts from JavaScript files.

Do	Don't
<link href="cdn.company.com/css/main.css" /> <script src="cdn.company.com/js/main.js"></script> Allows for CSP headers to deny all usage of inline and eval statements.	 XSS <script> window.onload = eval("alert(1)"); </script>

If inline scripts ***MUST*** be utilized, then you must implement Content-Security-Policy (CSP) hash or nonce. A CSP hash is preferred since the resource can be cached and can be configured on the web server. A CSP nonce must be dynamically generated per request, thus requires server-side code to set the header value, but doesn't need to compute the hash for each resource instance.

Nonce

```
<link href="cdn.company.com/css/main.css" nonce="5e5v90q" />
Content-Security-Policy: style-src 'nonce-5e5v90q'
```

Hash

```
<script src="cdn.company.com/css/main.js"></script>
Content-Security-Policy: script-src 'sha256-v90qNGx6fS48N0tRxiGkqveZETq72KgDVJCp2TC'
```

<frame> and <iFrame>

Avoid using frames and iFrames completely as they are easy attack vectors for click-jacking and XSS.

Do	Don't
Content-Security-Policy: frame-ancestor 'none' X-Fame-Options: DENY Frame-ancestor can be set to 'self' or an explicit source list. Both security headers impact <embed> and <object>, thus you may need to explicitly allow it such as embedded videos from YouTube: Content-Security-Policy: frame-ancestor 'self' https://www.yoututbe.com	<iframe src="https://www.example.com" ></iframe> It is impossible to know what script will be executed upon the loading of this HTML page.

Input Validations

Always check user inputs for valid values to avoid SQL injection attacks.

Do	Don't
<form> <input type="number" name="UID" onchange="sanitize( )" /> </form> <script> function sanitize(e) { . . . } </script>	SELECT * FROM Users WHERE UID = @userId <form> <input type="number" name="UID" value="1 or 1=1" /> </form> This value will cause the SQL query to morph into:

Sanitize the input value by: • Checking data type (e.g., integer) • Checking data range (e.g., 1 to 10) • Removing SQL injection text patterns (e.g., " ; DROP TABLE ABC -- ") • Escaping input text (e.g., " 'UID'; DROP TABLE ABC ") • Use sanitization libraries • Avoid dynamic SQL queries	SELECT * FROM Users WHERE UID = 1 or 1 = 1 Which will ***allow*** all users to be exposed.

Submitting Data

Always use an anti-forgery token to prevent malicious code from making malicious requests on behalf of a valid active session (CSRF or XSRF).

Do	Don't
<form> <input type="password" name="pw" value="" /> <input type="hidden" name="csrf_token" value="a46s" /> </form> The server will send a Set-Cookie (Path=/; Secure; SameSite=Strict) HTTP response header with the CSRF Token to use or dynamically altered the HTML to contain the token. The web client should then relay the save token value back to the server as a custom header (i.e., X-CSRF-Token) through an AJAX request. The Same-Origin policy prevents JavaScript from adding custom headers in any cross-origin requests, thus preventing CSRF.	<form> <input type="password" name="pw" value="" /> </form>

Web Cookies

All cookie should be as restrictive as possible and configured with the Secure flag to prevent XSS attacks. Setting SameSite to Strict is also recommended to combat CSRF.

Cookie Settings	Value	Note
cookie-name	string	Any ASCII except whitespace and special characters. Special prefixes __Secure-<cookie-name> must have: • **Secure** flag • Origin was HTTPS __Host-<cookie-name> must have: • **Secure** flag • Origin was HTTPS • **Domain** NOT set • **Path**=/
Domain	string	Setting the Domain will allow the cookie to have access to the host and all subdomains of host. Not setting it will restrict cookie to just the host of the current document URL WITHOUT access to any subdomains.
Path	string	Specifies the path that browser can send cookie from
Secure		Enforce cookie to be sent only through HTTPS
HttpOnly		Don't require being accessed from JavaScript
SameSite	Strict Lax	Allows cookie to be sent in cross-origin requests. • Strict: Only send cookie through direct navigation

	None	• Lax: Default; no cross-site request except during navigation from your site to another site • None: Both Cross-site and same-site. Must have Secure flag as well
Expires	Date	Absolute expiration date based on client and not server. Format: day-name, DD MMM YYYY HH:MM:SS GMT Day-name and months are case-sensitive. If omitted, the cookie becomes a session cookie that won't end until the web client is closed.
Max-Age	Seconds	Relative expiration date 0 or negative number will expire cookie immediately. Max-Age has precedence over Expires If omitted, the cookie becomes a session cookie that won't end until the web client is closed.
Partitioned		Cookie is stored using partitioned storage (Partitioned cookie) that is only tied to top-level site and only accessible from there Must have: • Secure flag • Path=/ • Use __Host prefix

Resources

- https://infosec.mozilla.org/guidelines/web_security
- https://content-security-policy.com/
- https://owasp.org/Top10/
- https://observatory.mozilla.org/
- https://SecurityHeaders.com