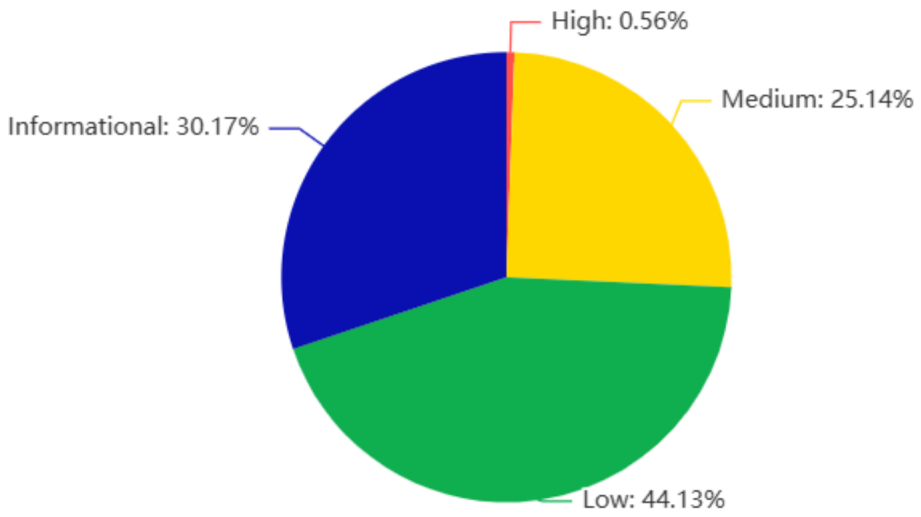
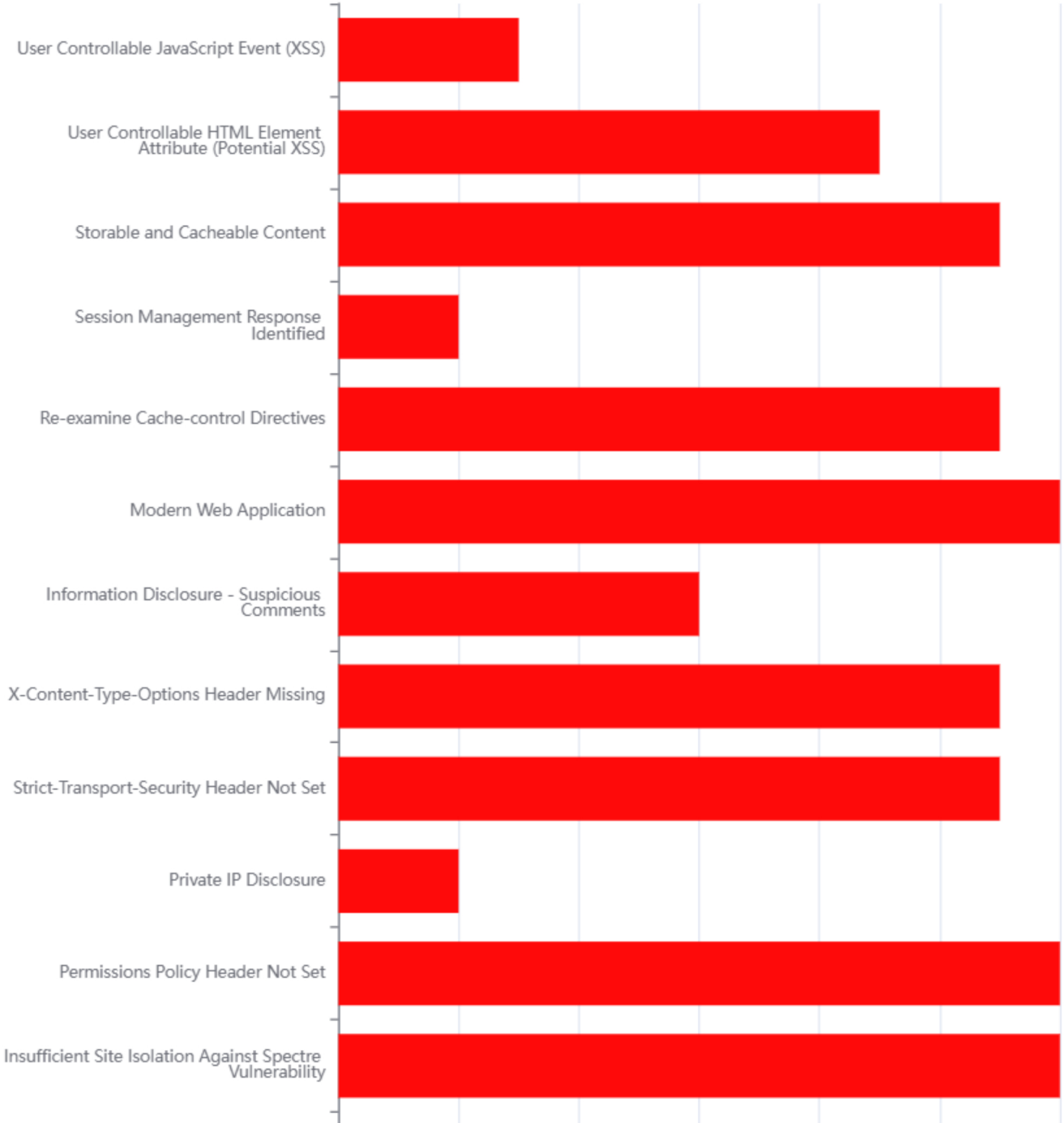
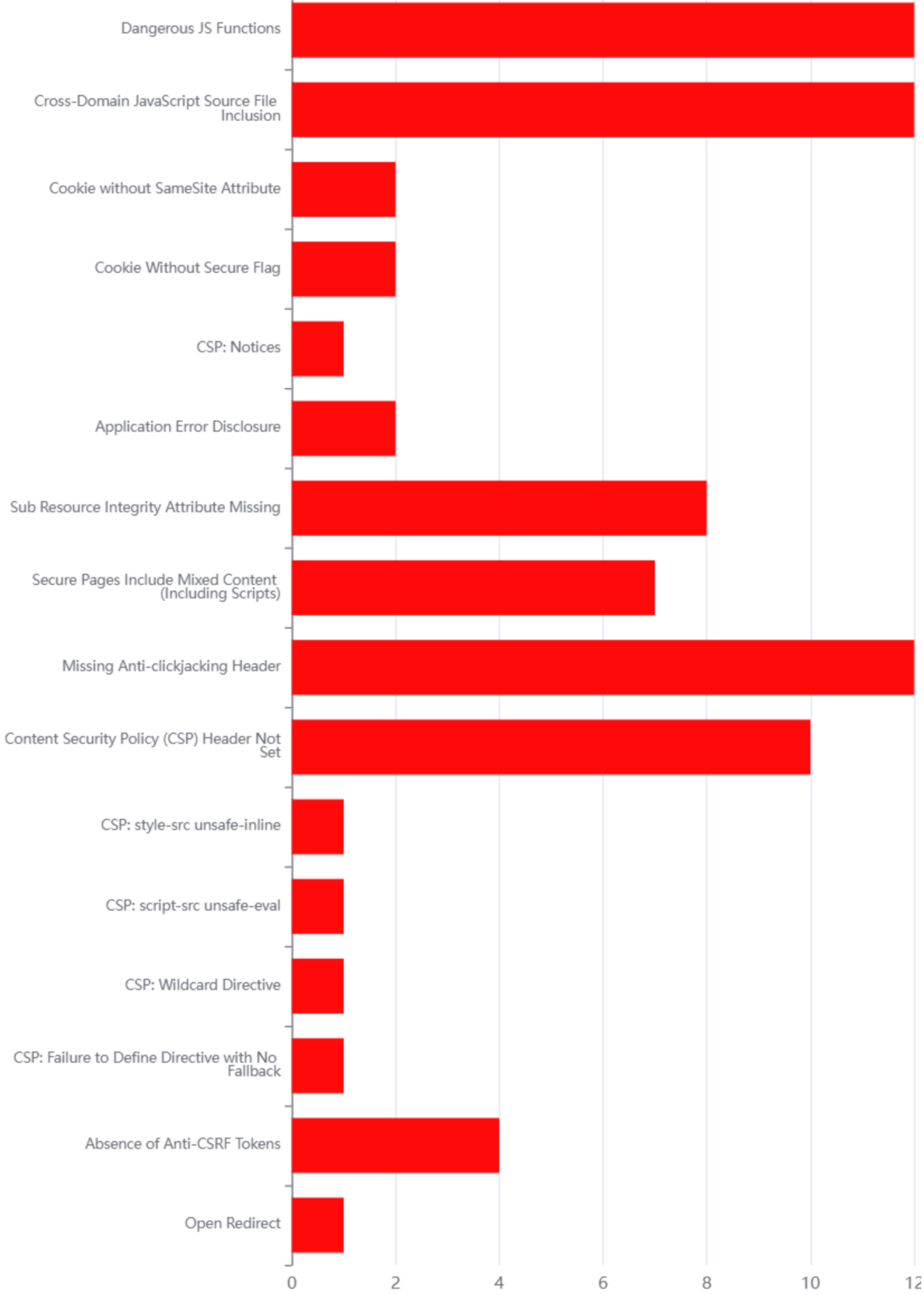
# Security Test Report

## Vulnerability Severity




## Vulnerability Count






# AI Summary 1/2:

## Report Metadata

- **Generated On:** Thu, 3 Apr 2025 20:57:29
- **Scanned Site:** https://public-firing-range.appspot.com, public-firing-range.appspot.com
- **Total Vulnerabilities Identified:** 20

### Top 3 Critical Vulnerabilities

## Top 3 Critical Vulnerabilities

### Open Redirect

- **Risk Level:** High
- **Count:** 1
- **Exploitability:** Medium
- **Impact:** High
- **Why is it critical?** Open redirects are one of the OWASP 2010 Top Ten vulnerabilities, allowing attackers to control offsite redirects, which can be used for phishing attacks or spamming.
- **Mitigation Strategy:** Validate parameters of the application script/program before sending 302 HTTP code (redirect) to the client browser. Implement safe redirect functionality that only redirects to relative URI's, or a list of trusted domains.
- **References:** https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html, https://cwe.mitre.org/data/definitions/601.html

### Absence of Anti-CSRF Tokens

- **Risk Level:** Medium
- **Count:** 4
- **Exploitability:** Low
- **Impact:** Medium
- **Why is it critical?** The absence of Anti-CSRF tokens makes the application vulnerable to cross-site request forgery attacks, which can be used to perform actions as the victim.
- **Mitigation Strategy:** Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Ensure that your application is free of cross-site scripting issues.
- **References:** https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html, https://cwe.mitre.org/data/definitions/352.html

### CSP: Failure to Define Directive with No Fallback

- **Risk Level:** Medium
- **Count:** 1
- **Exploitability:** High
- **Impact:** Medium
- **Why is it critical?** The failure to define a directive with no fallback in the Content Security Policy (CSP) makes the application vulnerable to certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.
- **Mitigation Strategy:** Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.
- **References:** https://www.w3.org/TR/CSP/, https://caniuse.com/#search=content+security+policy, https://content-security-policy.com/, https://github.com/HtmlUnit/htmlunit-csp, https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resourc

## Remaining Vulnerabilities

### CSP: Wildcard Directive

CSP: script-src unsafe-eval

CSP: style-src unsafe-inline

Content Security Policy (CSP) Header Not Set

Missing Anti-clickjacking Header

Secure Pages Include Mixed Content (Including Scripts)

Sub Resource Integrity Attribute Missing

Application Error Disclosure

CSP: Notices

Cookie Without Secure Flag

Cookie without SameSite Attribute

Cross-Domain JavaScript Source File Inclusion

Dangerous JS Functions

Insufficient Site Isolation Against Spectre Vulnerability

Permissions Policy Header Not Set

Private IP Disclosure

Strict-Transport-Security Header Not Set

### Additional Security Recommendations

- Identify recurring patterns in vulnerabilities and suggest general security improvements.
- Highlight missing security controls (e.g., weak authentication, lack of encryption, missing headers).
- Suggest proactive defense strategies.

### Final Risk Assessment

- **Overall Security Rating:** Medium
- **Key Areas of Concern:** Open Redirect, Absence of Anti-CSRF Tokens, CSP: Failure to Define Directive with No Fallback
- **Critical Fixes Required:** Open Redirect, Absence of Anti-CSRF Tokens, CSP: Failure to Define Directive with No Fallback
- **Suggested Next Steps:** Implement safe redirect functionality, use a vetted library or framework to avoid Anti-CSRF weaknesses, ensure proper configuration of Content-Security-Policy header.

## AI Summary 2/2:

### Report Metadata

- **Generated On:** Thu, 3 Apr 2025 20:57:29
- **Scanned Site:** https://public-firing-range.appspot.com
- **Total Vulnerabilities Identified:** 7

### Top 3 Critical Vulnerabilities

Storable and Cacheable Content

Storable and Cacheable Content

- **Risk Level:** Informational (Medium)
- **Count:** 11
- **Exploitability:** Medium
- **Impact:** Sensitive information may be leaked through caching components.
- **Why is it critical?** It affects user privacy and session security.
- **Mitigation Strategy:** Use HTTP response headers to limit caching: Cache-Control: no-cache, no-store, must-revalidate, private; Pragma: no-cache; Expires: 0.
- **References:** https://datatracker.ietf.org/doc/html/rfc7234, https://datatracker.ietf.org/doc/html/rfc7231, https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

### User Controllable HTML Element Attribute (Potential XSS)

- **Risk Level:** Informational (Low)
- **Count:** 9
- **Exploitability:** Low
- **Impact:** Potential XSS vulnerability through user-controlled HTML attribute values.
- **Why is it critical?** It can lead to cross-site scripting attacks.
- **Mitigation Strategy:** Validate all input and sanitize output before writing to any HTML attributes.
- **References:** https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

### User Controllable JavaScript Event (XSS)

- **Risk Level:** Informational (Low)
- **Count:** 3
- **Exploitability:** Low
- **Impact:** Potential XSS vulnerability through user-controlled JavaScript events.
- **Why is it critical?** It can lead to cross-site scripting attacks.
- **Mitigation Strategy:** Validate all input and sanitize output before writing to any Javascript on* events.
- **References:** https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

## Remaining Vulnerabilities

### Modern Web Application

### Re-examine Cache-control Directives

### Session Management Response Identified

## Additional Security Recommendations

- **Recurring Patterns:** Lack of proper cache control and potential XSS vulnerabilities.
- **Missing Security Controls:** Proper input validation and output sanitization.
- **Proactive Defense Strategies:** Implement a web application firewall (WAF) and conduct regular security audits.

## Final Risk Assessment

- **Overall Security Rating:** Medium
- **Key Areas of Concern:** Storable and cacheable content, potential XSS vulnerabilities.

- **Critical Fixes Required:** Implement proper cache control and input validation.
- **Suggested Next Steps:** Conduct a thorough security audit and implement a web application firewall (WAF).

# Report

| DAST Test Configurations |
| --- |
| **program name:** ZAP |
| **version:** 2.16.0 |
| **generated:** Thu, 3 Apr 2025 20:57:29 |

| Site General Information |
| --- |
| **name:** https://public-firing-range.appspot.com |
| **host:** public-firing-range.appspot.com |
| **port:** 443 |
| **ssl:** true |

| Name | Impact |
| --- | --- |
| **Open Redirect**<br>Risk: High (Medium) | Open redirects are one of the OWASP 2010 Top Ten vulnerabilities. This check looks at user-supplied input in query string parameters and POST data to identify where open redirects might be possible. Open redirects occur when an application allows user-supplied input (e.g. https://nottrusted.com) to control an offsite redirect. This is generally a pretty accurate way to find where 301 or 302 redirects could be exploited by spammers or phishing attacks.<br>``<br>For example an attacker could supply a user with the following link: https://example.com/example.php?url=https://malicious.example.com.<br>**Possible Solution:**<br>To avoid the open redirect vulnerability, parameters of the application script/program must be validated before sending 302 HTTP code (redirect) to the client browser. Implement safe redirect functionality that only redirects to relative URI's, or a list of trusted domains<br>**References:** |

| | |
|---|---|
| | https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html<br><br>https://cwe.mitre.org/data/definitions/601.html |
| **Absence of Anti-CSRF Tokens**<br>Risk: Medium (Low) | No Anti-CSRF tokens were found in a HTML submission form.<br><br>A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.<br><br> `CSRF attacks are effective in a number of situations, including:` ` * The victim has an active session on the target site.` ` * The victim is authenticated via HTTP auth on the target site.` ` * The victim is on the same local network as the target site.` <br><br>CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.<br><br>**Possible Solution:**<br>Phase: Architecture and Design<br><br>Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.<br><br>For example, use anti-CSRF packages such as the OWASP CSRFGuard.<br><br> `Phase: Implementation` `Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.` <br><br>Phase: Architecture and Design |

| | |
|---|---|
| | Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).<br><br>Note that this can be bypassed using XSS.<br><br> `Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.` `Note that this can be bypassed using XSS.` <br><br>Use the ESAPI Session Management control.<br><br>This control includes a component for CSRF.<br><br> `Do not use the GET method for any request that triggers a state change.` <br><br>Phase: Implementation<br><br>Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.<br><br>**References:**<br>https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html<br><br>https://cwe.mitre.org/data/definitions/352.html |
| **CSP: Failure to Define Directive with No Fallback**<br>Risk: Medium (High) | The Content Security Policy fails to define one of the directives that has no fallback. Missing/excluding them is the same as allowing anything.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.<br><br>**References:**<br>https://www.w3.org/TR/CSP/<br><br>https://caniuse.com/#search=content+security+policy<br><br>https://content-security-policy.com/<br><br>https://github.com/HtmlUnit/htmlunit-csp |

| | |
|---|---|
| | https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources |
| **CSP: Wildcard Directive**<br>Risk: Medium (High) | Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page – covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.<br><br>**References:**<br>https://www.w3.org/TR/CSP/<br><br>https://caniuse.com/#search=content+security+policy<br><br>https://content-security-policy.com/<br><br>https://github.com/HtmlUnit/htmlunit-csp<br><br>https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources |
| **CSP: script-src unsafe-eval**<br>Risk: Medium (High) | Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page – covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.<br><br>**References:** |

| | |
|---|---|
| | https://www.w3.org/TR/CSP/<br><br>https://caniuse.com/#search=content+security+policy<br><br>https://content-security-policy.com/<br><br>https://github.com/HtmlUnit/htmlunit-csp<br><br>https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources |
| **CSP: style-src unsafe-inline**<br>Risk: Medium (High) | Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page – covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.<br><br>**References:**<br>https://www.w3.org/TR/CSP/<br><br>https://caniuse.com/#search=content+security+policy<br><br>https://content-security-policy.com/<br><br>https://github.com/HtmlUnit/htmlunit-csp<br><br>https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources |
| | Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page – covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java |

| | |
|---|---|
| **Content Security Policy (CSP) Header Not Set**<br>Risk: Medium (High) | applets, ActiveX, audio and video files.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.<br><br>**References:**<br>https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy<br><br>https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html<br><br>https://www.w3.org/TR/CSP/<br><br>https://w3c.github.io/webappsec-csp/<br><br>https://web.dev/articles/csp<br><br>https://caniuse.com/#feat=contentsecuritypolicy<br><br>https://content-security-policy.com/ |
| **Missing Anti-clickjacking Header**<br>Risk: Medium (Medium) | The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.<br><br>**Possible Solution:**<br>Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.<br><br>If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.<br><br>**References:**<br>https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options |
| **Secure Pages Include Mixed Content** | The page includes mixed content, that is content accessed via HTTP instead of HTTPS.<br><br>**Possible Solution:**<br>A page that is available over SSL/TLS must be comprised completely of content which is transmitted over SSL/TLS.<br><br>The page must not contain any content that is transmitted over |

| | |
|---|---|
| **(Including Scripts)**<br>Risk: Medium (Medium) | The page must not contain any content that is transmitted over unencrypted HTTP.<br><br>This includes content from third party sites.<br><br>**References:**<br>https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html |
| **Sub Resource Integrity Attribute Missing**<br>Risk: Medium (High) | The integrity attribute is missing on a script or link tag served by an external server. The integrity tag prevents an attacker who have gained access to this server from injecting a malicious content.<br><br>**Possible Solution:**<br>Provide a valid integrity attribute to the tag.<br><br>**References:**<br>https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity |
| **Application Error Disclosure** | This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page. |
| **CSP: Notices**<br>Risk: Low (High) | Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.<br><br>**References:**<br>https://www.w3.org/TR/CSP/<br><br>https://caniuse.com/#search=content+security+policy<br><br>https://content-security-policy.com/ |

| | |
|---|---|
| | https://github.com/HtmlUnit/htmlunit-csp<br><br>https://developers.google.com/web/fundamentals/security/csp#policy_applies_to_a_wide_variety_of_resources |
| **Cookie Without Secure Flag**<br>Risk: Low (Medium) | A cookie has been set without the secure flag, which means that the cookie can be accessed via unencrypted connections.<br><br>**Possible Solution:**<br>Whenever a cookie contains sensitive information or is a session token, then it should always be passed using an encrypted channel. Ensure that the secure flag is set for cookies containing such sensitive information.<br><br>**References:**<br>https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html |
| **Cookie without SameSite Attribute**<br>Risk: Low (Medium) | A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.<br><br>**Possible Solution:**<br>Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.<br><br>**References:**<br>https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site |
| **Cross-Domain JavaScript Source File Inclusion** | The page includes one or more script files from a third-party domain. |
| **Dangerous JS Functions**<br>Risk: Low (Low) | A dangerous JS function seems to be in use that would leave the site vulnerable.<br><br>**Possible Solution:**<br>See the references for security advice on the use of these functions.<br><br>**References:**<br>https://angular.io/guide/security |

| | |
|---|---|
| **Insufficient Site Isolation Against Spectre Vulnerability**<br>Risk: Low (Medium) | Cross-Origin-Resource-Policy header is an opt-in header designed to counter side-channels attacks like Spectre. Resource should be specifically set as shareable amongst different origins.<br><br>**Possible Solution:**<br>Ensure that the application/web server sets the Cross-Origin-Resource-Policy header appropriately, and that it sets the Cross-Origin-Resource-Policy header to 'same-origin' for all web pages.<br><br>'same-site' is considered as less secured and should be avoided.<br><br>If resources must be shared, set the header to 'cross-origin'.<br><br>If possible, ensure that the end user uses a standards-compliant and modern web browser that supports the Cross-Origin-Resource-Policy header (https://caniuse.com/mdn-http_headers_cross-origin-resource-policy).<br><br>**References:**<br>https://developer.mozilla.org/en-US/docs/Web/HTTP/Cross-Origin_Resource_Policy |
| **Permissions Policy Header Not Set**<br>Risk: Low (Medium) | Permissions Policy Header is an added layer of security that helps to restrict from unauthorized access or usage of browser/client features by web resources. This policy ensures the user privacy by limiting or specifying the features of the browsers can be used by the web resources. Permissions Policy provides a set of standard HTTP headers that allow website owners to limit which features of browsers can be used by the page such as camera, microphone, location, full screen etc.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is configured to set the Permissions-Policy header.<br><br>**References:**<br>https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy<br><br>https://developer.chrome.com/blog/feature-policy/<br><br>https://scotthelme.co.uk/a-new-security-header-feature-policy/<br><br>https://w3c.github.io/webappsec-feature-policy/<br><br>https://www.smashingmagazine.com/2018/12/feature-policy/ |
| | A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been |

| | |
|---|---|
| **Private IP Disclosure**<br>Risk: Low (Medium) | Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.<br><br>**Possible Solution:**<br>Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.<br><br>**References:**<br>https://tools.ietf.org/html/rfc1918 |
| **Strict-Transport-Security Header Not Set**<br>Risk: Low (High) | HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL). HSTS is an IETF standards track protocol and is specified in RFC 6797.<br><br>**Possible Solution:**<br>Ensure that your web server, application server, load balancer, etc. is configured to enforce Strict-Transport-Security.<br><br>**References:**<br>https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html<br><br>https://owasp.org/www-community/Security_Headers<br><br>https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security<br><br>https://caniuse.com/stricttransportsecurity<br><br>https://datatracker.ietf.org/doc/html/rfc6797 |
| **X-Content-Type-Options Header Missing**<br>Risk: Low (Medium) | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.<br><br>**Possible Solution:**<br>Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.<br><br>If possible, ensure that the end user uses a standards-compliant |

| | |
|---|---|
| (Medium) | and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.<br><br>**References:**<br>https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85)<br><br>https://owasp.org/www-community/Security_Headers |
| **Information Disclosure - Suspicious Comments** | The response appears to contain suspicious comments which may help an attacker.  |
| **Modern Web Application** | The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.  |
| **Re-examine Cache-control Directives**<br>Risk: Informational (Low) | The cache-control header has not been set properly or is missing, allowing the browser and proxies to cache content. For static assets like css, js, or image files this might be intended, however, the resources should be reviewed to ensure that no sensitive content will be cached.<br><br>**Possible Solution:**<br>For secure content, ensure the cache-control HTTP header is set with "no-cache, no-store, must-revalidate". If an asset should be cached consider setting the directives "public, max-age, immutable".<br><br>**References:**<br>https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#web-content-caching<br><br>https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control<br><br>https://grayduck.mn/2021/09/13/cache-control-recommendations/ |
| **Session Management Response** | The given response has been identified as containing a session management token. The 'Other Info' field contains a set of header tokens that can be used in the Header Based Session Management Method. If the request is in a context which has a Session Management Method set to "Auto-Detect" then this rule will change the session management to use the tokens identified. |

| | |
|---|---|
| **Identified**<br>Risk: Informational (Medium) | **Possible Solution:**<br>This is an informational alert rather than a vulnerability and so there is nothing to fix.<br><br>**References:**<br>https://www.zaproxy.org/docs/desktop/addons/authentication-helper/session-mgmt-id |
| **Storable and Cacheable Content**<br>Risk: Informational (Medium) | The response contents are storable by caching components such as proxy servers, and may be retrieved directly from the cache, rather than from the origin server by the caching servers, in response to similar requests from other users. If the response data is sensitive, personal or user-specific, this may result in sensitive information being leaked. In some cases, this may even result in a user gaining complete control of the session of another user, depending on the configuration of the caching components in use in their environment. This is primarily an issue where "shared" caching servers such as "proxy" caches are configured on the local network. This configuration is typically found in corporate or educational environments, for instance.<br><br>**Possible Solution:**<br>Validate that the response does not contain sensitive, personal or user-specific information. If it does, consider the use of the following HTTP response headers, to limit, or prevent the content being stored and retrieved from the cache by another user:<br><br>Cache-Control: no-cache, no-store, must-revalidate, private<br><br>Pragma: no-cache<br><br>Expires: 0<br><br>This configuration directs both HTTP 1.0 and HTTP 1.1 compliant caching servers to not store the response, and to not retrieve the response (without validation) from the cache, in response to a similar request.<br><br>**References:**<br>https://datatracker.ietf.org/doc/html/rfc7234<br><br>https://datatracker.ietf.org/doc/html/rfc7231<br><br>https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html |
| | This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for |

| | |
|---|---|
| **User Controllable HTML Element Attribute (Potential XSS)** Risk: Informational (Low) | XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.<br><br>**Possible Solution:**<br>Validate all input and sanitize output it before writing to any HTML attributes.<br><br>**References:**<br>https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html |
| **User Controllable JavaScript Event (XSS)** Risk: Informational (Low) | This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.<br><br>**Possible Solution:**<br>Validate all input and sanitize output it before writing to any Javascript on* events.<br><br>**References:**<br>https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html |