















|  |  |  |
| --- | --- | --- |
| **Alert** | **Info** | **Solution** |
| SQL injection may be possible | The page results were successfully manipulated using the boolean conditions [ZAP' AND '1'='1' -- ] and [ZAP' AND '1'='2' -- ]  The parameter value being modified was stripped from the HTML output for the purposes of the comparison  Data was returned for the original parameter.  The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter | Do not trust client side input, even if there is client side validation in place.  In general, type check all data on the server side.  If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'  If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.  If database Stored Procedures can be used, use them.  Do \*not\* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!  Do not create dynamic SQL queries using simple string concatenation.  Escape all data received from the client.  Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input.  Apply the principle of least privilege by using the least privileged database user possible.  In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.  Grant the minimum database access that is necessary for the application. |
| SQL injection may be possible | The page results were successfully manipulated using the boolean conditions [Login AND 1=1 -- ] and [Login AND 1=2 -- ]  The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison  Data was returned for the original parameter.  The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter | Do not trust client side input, even if there is client side validation in place.  In general, type check all data on the server side.  If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'  If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.  If database Stored Procedures can be used, use them.  Do \*not\* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!  Do not create dynamic SQL queries using simple string concatenation.  Escape all data received from the client.  Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input.  Apply the principle of least privilege by using the least privileged database user possible.  In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.  Grant the minimum database access that is necessary for the application. |
| SQL injection may be possible | The page results were successfully manipulated using the boolean conditions [Sign Up' AND '1'='1' -- ] and [Sign Up' AND '1'='2' -- ]  The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison  Data was returned for the original parameter.  The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter | Do not trust client side input, even if there is client side validation in place.  In general, type check all data on the server side.  If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'  If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.  If database Stored Procedures can be used, use them.  Do \*not\* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!  Do not create dynamic SQL queries using simple string concatenation.  Escape all data received from the client.  Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input.  Apply the principle of least privilege by using the least privileged database user possible.  In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.  Grant the minimum database access that is necessary for the application. |
| X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks | * <https://dat-250-test4.herokuapp.com/> * <https://dat-250-test4.herokuapp.com/index.html> * <https://dat-250-test4.herokuapp.com/login_page.html> * <https://dat-250-test4.herokuapp.com/register.html> * Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options | Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (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. ALLOW-FROM allows specific websites to frame the web page in supported web browsers). |
| 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. | Reference : <https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site>   * <https://dat-250-test4.herokuapp.com/> * <https://dat-250-test4.herokuapp.com/index.html> * <https://dat-250-test4.herokuapp.com/login_page.html> * <https://dat-250-test4.herokuapp.com/register.html> | Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies |
| A cookie has been set without the secure flag, which means that the cookie can be accessed via unencrypted connections. | Reference: <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>   * <https://dat-250-test4.herokuapp.com/> * <https://dat-250-test4.herokuapp.com/index.html> * <https://dat-250-test4.herokuapp.com/login_page.html> * <https://dat-250-test4.herokuapp.com/register.html> | 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. |
| The cache-control and pragma HTTP header have not been set properly or are missing allowing the browser and proxies to cache content. | * <https://dat-250-test4.herokuapp.com/> * <https://dat-250-test4.herokuapp.com/index.html> * <https://dat-250-test4.herokuapp.com/login_page.html> * <https://dat-250-test4.herokuapp.com/register.html> * <https://dat-250-test4.herokuapp.com/static/style.css> | Whenever possible ensure the cache-control HTTP header is set with no-cache, no-store, must-revalidate; and that the pragma HTTP header is set with no-cache. |
| 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. | * <https://dat-250-test4.herokuapp.com/> * <https://dat-250-test4.herokuapp.com/index.html> * <https://dat-250-test4.herokuapp.com/login_page.html> * <https://dat-250-test4.herokuapp.com/register.html> * <https://dat-250-test4.herokuapp.com/static/style.css>   This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.  At "High" threshold this scanner will not alert on client or server error responses. | 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.  If possible, ensure that the end user uses a standards-compliant 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. |