ZAP Scanning Report WebGoat

Site: http://localhost:8080

Generated on Wed, 19 Jan 2022 15:47:54

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	1
Low	4
Informational	1

Alerts

Name	Risk Level	Number of Instances
SQL Injection	High	1
Parameter Tampering	Medium	1
Absence of Anti-CSRF Tokens	Low	5
Cookie No HttpOnly Flag	Low	1
Cookie without SameSite Attribute	Low	1
Timestamp Disclosure - Unix	Low	4
Loosely Scoped Cookie	Informational	1

Alert Detail

High	SQL Injection
Description	SQL injection may be possible.
URL	http://localhost:8080/WebGoat/register.mvc
Method	POST
Attack	agree AND 1=1 --
Evidence	
Instances	1
	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.

Solution	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 an 'allow list' of allowed characters, or a 'deny list' 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.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
CWE Id	89
WASC Id	19
Plugin Id	40018

Medium	**Parameter Tampering**
Description	Parameter manipulation caused an error page or Java stack trace to be displayed. This indicated lack of exception handling and potential areas for further exploit.
URL	http://localhost:8080/WebGoat/register.mvc
Method	POST
Attack	
Evidence	javax.servlet.http.HttpServlet.service(HttpServlet.java:523)\n\tat
Instances	1
Solution	Identify the cause of the error and fix it. Do not trust client side input and enforce a tight check in the server side. Besides, catch the exception properly. Use a generic 500 error page for internal server error.
Reference	
CWE Id	472
WASC Id	20
Plugin Id	40008

Low	**Absence of Anti-CSRF Tokens**
Description	No Anti-CSRF tokens were found in a HTML submission form. 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. 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.

	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.
URL	http://localhost:8080/WebGoat
Method	GET
Attack	
Evidence	<form action="/WebGoat/login" method='POST' style="width: 200px;">
URL	http://localhost:8080/WebGoat/login
Method	GET
Attack	
Evidence	<form action="/WebGoat/login" method='POST' style="width: 200px;">
URL	http://localhost:8080/WebGoat/login?error
Method	GET
Attack	
Evidence	<form action="/WebGoat/login" method='POST' style="width: 200px;">
URL	http://localhost:8080/WebGoat/registration
Method	GET
Attack	
Evidence	<form class="form-horizontal" action="/WebGoat/register.mvc" method='POST'>
URL	http://localhost:8080/WebGoat/register.mvc
Method	POST
Attack	
Evidence	<form class="form-horizontal" action="/WebGoat/register.mvc" method='POST'>
Instances	5
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. 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). Note that this can be bypassed using XSS. 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. Use the ESAPI Session Management control.

	This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation 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.
Reference	http://projects.webappsec.org/Cross-Site-Request-Forgery http://cwe.mitre.org/data/definitions/352.html
CWE Id	352
WASC Id	9
Plugin Id	10202

Low	**Cookie No HttpOnly Flag**
Description	A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
URL	http://localhost:8080/WebGoat/
Method	GET
Attack	
Evidence	Set-Cookie: JSESSIONID
Instances	1
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	https://owasp.org/www-community/HttpOnly
CWE Id	1004
WASC Id	13
Plugin Id	10010

Low	**Cookie without SameSite Attribute**
Description	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.
URL	http://localhost:8080/WebGoat/
Method	GET
Attack	
Evidence	Set-Cookie: JSESSIONID
Instances	1
Solution	Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.
Reference	https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site
CWE Id	1275
WASC Id	13
Plugin Id	10054

Low	**Timestamp Disclosure - Unix**
Description	A timestamp was disclosed by the application/web server - Unix
URL	http://localhost:8080/WebGoat/plugins/bootstrap/css/bootstrap.min.css
Method	GET

Attack	
Evidence	33333333
URL	http://localhost:8080/WebGoat/plugins/bootstrap/css/bootstrap.min.css
Method	GET
Attack	
Evidence	42857143
URL	http://localhost:8080/WebGoat/plugins/bootstrap/css/bootstrap.min.css
Method	GET
Attack	
Evidence	66666667
URL	http://localhost:8080/WebGoat/plugins/bootstrap/css/bootstrap.min.css
Method	GET
Attack	
Evidence	80000000
Instances	4
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage
CWE Id	200
WASC Id	13
Plugin Id	10096

Informational	**Loosely Scoped Cookie**
Description	Cookies can be scoped by domain or path. This check is only concerned with domain scope. The domain scope applied to a cookie determines which domains can access it. For example, a cookie can be scoped strictly to a subdomain e.g. www.nottrusted.com, or loosely scoped to a parent domain e.g. nottrusted.com. In the latter case, any subdomain of nottrusted.com can access the cookie. Loosely scoped cookies are common in mega-applications like google.com and live.com. Cookies set from a subdomain like app.foo.bar are transmitted only to that domain by the browser. However, cookies scoped to a parent-level domain may be transmitted to the parent, or any subdomain of the parent.
URL	http://localhost:8080/WebGoat/
Method	GET
Attack	
Evidence	
Instances	1
Solution	Always scope cookies to a FQDN (Fully Qualified Domain Name).
Reference	https://tools.ietf.org/html/rfc6265#section-4.1 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 http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy_for_cookies
CWE Id	565
WASC Id	15
Plugin Id	90033