Security Scan Report - DVWA

Scan Date: 2025-03-04 11:06:06

Executive Summary

This report summarizes a security scan of the GitHub repository: https://github.com/digininja/DVWA.git. We found 74 potential security issues. Breakdown: 6 Critical, 68 High, 0 Medium, 0 Low, 0 Informational issues.

Severity	Count	Risk Level
Critical	6	IMMEDIATE ACTION REQUIRED
High	68	URGENT ACTION NEEDED
Medium	0	
Low	0	
Info	0	

Top Critical Issues

1. **Use of eval() - Potential code injection risk** in *dvwa/js/dvwaPage.js* (line 7) - Critical
2. **Use of eval() - Potential code injection risk** in *vulnerabilities/view_help.php* (line 20) - Critical
3. **Use of eval() - Potential code injection risk** in *vulnerabilities/view_help.php* (line 22) - Critical

Overall Risk Assessment

Risk Score: 81.6/100 (Critical Risk)

Vulnerability Distribution

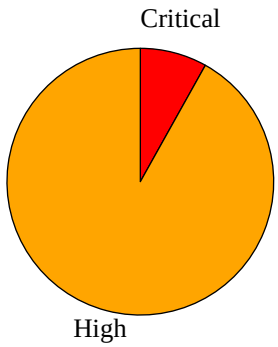

Figure 1: Distribution of vulnerabilities by severity

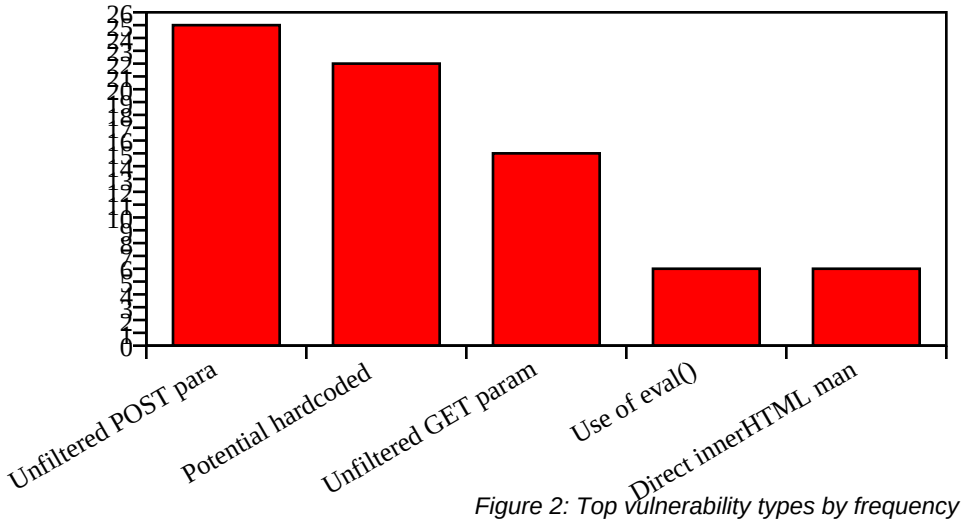

Figure 2: Top vulnerability types by frequency

What We Found

Critical Severity Issues (6)

File	Line	Issue	Context
dvwa/js/dvwaPage.js	7	Use of eval() - Potential code injection risk	`//eval("page" + id + " = window.open(URL, '" + id + "', 'too...`
Remediation:		Replace eval() with safer alternatives like JSON.parse() for JSON data or dedicated parsers for specific formats.	

vulnerabilities/view_help.php	20	Use of eval() - Potential code injection risk	`eval( '?>' . file_get_contents ( DVWA_WEB_PAGE_TO_ROOT . " vul...`
Remediation:		Replace eval() with safer alternatives like JSON.parse() for JSON data or dedicated parsers for specific formats.	
vulnerabilities/view_help.php	22	Use of eval() - Potential code injection risk	`eval( '?>' . file_get_contents ( DVWA_WEB_PAGE_TO_ROOT . " vul...`
Remediation:		Replace eval() with safer alternatives like JSON.parse() for JSON data or dedicated parsers for specific formats.	
vulnerabilities/javascript/source/high_unobfuscated.js	83	Use of eval() - Potential code injection risk	`var crypto = eval("require('crypto')");`
Remediation:		Replace eval() with safer alternatives like JSON.parse() for JSON data or dedicated parsers for specific formats.	
vulnerabilities/javascript/source/high_unobfuscated.js	84	Use of eval() - Potential code injection risk	`var Buffer = eval("require('buffer').Buffer");`
Remediation:		Replace eval() with safer alternatives like JSON.parse() for JSON data or dedicated parsers for specific formats.	
vulnerabilities/javascript/source/high.js	1	Use of eval() - Potential code injection risk	`var a=['fromCharCode','toString','replace','BeJ','\x5cw+','L...`
Remediation:		Replace eval() with safer alternatives like JSON.parse() for JSON data or dedicated parsers for specific formats.	

High Severity Issues (68)

File	Line	Issue	Context
security.php	13	Unfiltered POST parameter - Injection risk	`if( isset( $_POST['seclev_submit'] ) ) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
login.php	39	Potential hardcoded credential	`$query = "SELECT * FROM `users` WHERE user='$user' AND pass...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
external/recaptcha/recaptchalib.php	41	Potential hardcoded credential	`  <div class='g-recaptcha' data-theme='dark' data-sitek...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
dvwa/includes/dvwaPage.inc.php	108	Unfiltered POST parameter - Injection risk	`if (array_key_exists ("Login", $_POST) && $_POST['Login'] ==...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/brute/source/medium.php	14	Potential hardcoded credential	`$query = "SELECT * FROM `users` WHERE user = '$user' AND pa...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	

vulnerabilities/sqli/test.php	4	Potential hardcoded credential	`$password = "password";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/brute/source/high.php	19	Potential hardcoded credential	`$query = "SELECT * FROM `users` WHERE user = '$user' AND pa...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/brute/source/low.php	12	Potential hardcoded credential	`$query = "SELECT * FROM `users` WHERE user = '$user' AND pa...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/ecb_attack.php	17	Potential hardcoded credential	`$key = "ik ben een aardbei";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/xor_theory.php	19	Potential hardcoded credential	`$key = "wachtwoord";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/token_library_impossible.php	40	Potential hardcoded credential	`$token = "userid:2";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/token_library_high.php	37	Potential hardcoded credential	`$token = "userid:2";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/low.php	16	Potential hardcoded credential	`$key = "wachtwoord";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/low.php	29	Unfiltered POST parameter - Injection risk	`$message = $_POST['message'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/cryptography/source/low.php	30	Unfiltered POST parameter - Injection risk	`if (array_key_exists ('direction', $_POST) && $_POST['direct...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/cryptography/source/low.php	39	Unfiltered POST parameter - Injection risk	`$password = $_POST['password'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/brute/source/impossible.php	3	Unfiltered POST parameter - Injection risk	`if( isset( $_POST[ 'Login' ] ) && isset ($_POST['username']). ..`

Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/info.php	17	Unfiltered GET parameter - Injection risk	`if (array_key_exists ("id", $_GET) && is_numeric($_GET['id' ]...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/info.php	18	Unfiltered GET parameter - Injection risk	`switch (intval ($_GET['id'])) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/medium.php	3	Unfiltered GET parameter - Injection risk	`if (array_key_exists ("redirect", $_GET) && $_GET['redirect'...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/medium.php	4	Unfiltered GET parameter - Injection risk	`if (preg_match ("/http:\/\/|ht tps:\/\//i", $_GET['redirect']. ..`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/medium.php	11	Unfiltered GET parameter - Injection risk	`header ("location: " . $_GET['redirect']);`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/low.php	3	Unfiltered GET parameter - Injection risk	`if (array_key_exists ("redirect", $_GET) && $_GET['redirect'...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/low.php	4	Unfiltered GET parameter - Injection risk	`header ("location: " . $_GET['redirect']);`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/impossible.php	5	Unfiltered GET parameter - Injection risk	`if (array_key_exists ("redirect", $_GET) && is_numeric($_GET...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/impossible.php	6	Unfiltered GET parameter - Injection risk	`switch (intval ($_GET['redirec t'])) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/authbypass/authbypass.js	43	Direct innerHTML manipulation - XSS risk	`cell0.innerHTML = user['user_id'] + '<input type="hidden" id...`
Remediation:		Use proper output encoding and content security policies. Consider using template systems that auto-escape output.	
vulnerabilities/authbypass/authbypass.js	45	Direct innerHTML manipulation - XSS risk	`cell1.innerHTML = '<input type="text" id="first_name_' + use...`
Remediation:		Use proper output encoding and content security policies. Consider using template systems that auto-escape output.	

vulnerabilities/authbypass/authbypass.js	47	Direct innerHTML manipulation - XSS risk	`cell2.innerHTML = '<input type="text" id="surname_' + user['...`
Remediation:		Use proper output encoding and content security policies. Consider using template systems that auto-escape output.	
vulnerabilities/authbypass/authbypass.js	49	Direct innerHTML manipulation - XSS risk	`cell3.innerHTML = '<input type="button" value="Update" oncli...`
Remediation:		Use proper output encoding and content security policies. Consider using template systems that auto-escape output.	
vulnerabilities/cryptography/source/medium.php	10	Potential hardcoded credential	`$key = "ik ben een aardbei";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/cryptography/source/medium.php	21	Unfiltered POST parameter - Injection risk	`$token = $_POST['token'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/captcha/source/impossible.php	29	Unfiltered POST parameter - Injection risk	`$_POST['g-recaptcha-response']`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/high.php	3	Unfiltered GET parameter - Injection risk	`if (array_key_exists ("redirect", $_GET) && $_GET['redirect'...`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/high.php	4	Unfiltered GET parameter - Injection risk	`if (strpos($_GET['redirect'], "info.php") !== false) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/open_redirect/source/high.php	5	Unfiltered GET parameter - Injection risk	`header ("location: " . $_GET['redirect']);`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/captcha/source/high.php	14	Unfiltered POST parameter - Injection risk	`$_POST['g-recaptcha-response']`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/captcha/source/high.php	30	Potential hardcoded credential	`$insert = "UPDATE `users` SET password = '$pass_new' WHERE u...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/captcha/source/low.php	14	Unfiltered POST parameter - Injection risk	`$_POST['g-recaptcha-response']`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/captcha/source/low.php	60	Potential hardcoded credential	`$insert = "UPDATE `users` SET password = '$pass_new' WHERE u...`

Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/captcha/source/medium.php	14	Unfiltered POST parameter - Injection risk	`$_POST['g-recaptcha-response']`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/captcha/source/medium.php	68	Potential hardcoded credential	`$insert = "UPDATE `users` SET password = '$pass_new' WHERE u...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/csrf/source/medium.php	18	Potential hardcoded credential	`$insert = "UPDATE `users` SET password = '$pass_new' WHERE u...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/csrf/source/low.php	16	Potential hardcoded credential	`$insert = "UPDATE `users` SET password = '$pass_new' WHERE u...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/csrf/source/high.php	43	Potential hardcoded credential	`$insert = "UPDATE `users` SET password = '" . $pass_new . " '...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/csrf/test_credentials.php	21	Potential hardcoded credential	`$query = "SELECT * FROM `users` WHERE user='$user' AND pass...`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/xss_d/source/medium.php	5	Unfiltered GET parameter - Injection risk	`$default = $_GET['default'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/low.php	13	Unfiltered POST parameter - Injection risk	`if (isset ($_POST['include'])) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/low.php	15	Unfiltered POST parameter - Injection risk	`<script src='" . $_POST['include'] . " '></script>`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/medium.php	14	Unfiltered POST parameter - Injection risk	`if (isset ($_POST['include'])) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/medium.php	16	Unfiltered POST parameter - Injection risk	`" . $_POST['include'] . "`
Remediation:		Review the code and consider refactoring to follow security best practices.	

vulnerabilities/csp/source/high.js	9	Direct innerHTML manipulation - XSS risk	`document.getElementById("answer").innerHTML = obj['answer'];`
Remediation:		Use proper output encoding and content security policies. Consider using template systems that auto-escape output.	
vulnerabilities/csp/source/impossible.js	9	Direct innerHTML manipulation - XSS risk	`document.getElementById("answer").innerHTML = obj['answer'];`
Remediation:		Use proper output encoding and content security policies. Consider using template systems that auto-escape output.	
vulnerabilities/csp/source/high.php	8	Unfiltered POST parameter - Injection risk	`if (isset ($_POST['include'])) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/high.php	10	Unfiltered POST parameter - Injection risk	`" . $_POST['include'] . "`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/impossible.php	9	Unfiltered POST parameter - Injection risk	`if (isset ($_POST['include'])) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/impossible.php	11	Unfiltered POST parameter - Injection risk	`" . $_POST['include'] . "`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/xss_d/source/high.php	7	Unfiltered GET parameter - Injection risk	`switch ($_GET['default']) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/csp/source/jsonp.php	5	Unfiltered GET parameter - Injection risk	`$callback = $_GET['callback'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/api/src/Login.php	10	Potential hardcoded credential	`private const ACCESS_TOKEN_SECRET = "12345";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/api/src/Login.php	12	Potential hardcoded credential	`private const REFRESH_TOKEN_SECRET = "98765";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/api/src/LoginController.php	89	Unfiltered POST parameter - Injection risk	`switch ($_POST['grant_type']) {`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/api/src/LoginController.php	93	Unfiltered POST parameter - Injection risk	`$username = $_POST['username'];`

Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/api/src/LoginController.php	94	Unfiltered POST parameter - Injection risk	`$password = $_POST['password'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/api/src/LoginController.php	110	Unfiltered POST parameter - Injection risk	`$refresh_token = $_POST['refresh_token'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/api/src/Token.php	10	Potential hardcoded credential	`private const ENCRYPTION_KEY = "Paintbrush";`
Remediation:		Move sensitive data to environment variables or a secure vault/keystore service.	
vulnerabilities/javascript/index.php	37	Unfiltered POST parameter - Injection risk	`$phrase = $_POST['phrase'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	
vulnerabilities/javascript/index.php	38	Unfiltered POST parameter - Injection risk	`$token = $_POST['token'];`
Remediation:		Review the code and consider refactoring to follow security best practices.	

Remediation Guidance

This section provides general guidance for addressing the types of vulnerabilities found in your codebase. For specific remediation steps, refer to the individual findings in the detailed sections above.

• Replace dangerous functions like eval() with safer alternatives. Use JSON.parse() for JSON data or dedicated parsers for specific formats.
• Move sensitive data to environment variables or a secure vault/keystore service. Never commit credentials to version control.
• Use subprocess.run() with shell=False and pass arguments as a list instead of a string. Validate and sanitize all user inputs.
• Use proper output encoding and content security policies. Consider using template systems that auto-escape output.
• Use parameterized queries or an ORM instead of string concatenation for SQL queries. Never trust user input in database operations.
• Follow the principle of least privilege. Disable debug modes in production and ensure proper SSL verification.

Appendix: Scan Configuration

Repository URL: https://github.com/digininja/DVWA.git
Scan Date: 2025-03-04 11:06:06
Tool Version: GitHubScan 1.0
Scan Type: Static Code Analysis

Glossary of Security Terms

• **SAST (Static Application Security Testing):** Analysis of source code to identify security vulnerabilities without executing the program.
• **Code Injection:** A vulnerability that allows an attacker to insert and execute malicious code in an application.
• **Cross-Site Scripting (XSS):** A vulnerability that allows attackers to inject client-side scripts into web pages viewed by other users.
• **SQL Injection:** A code injection technique that exploits vulnerabilities in database-driven applications.
• **Hardcoded Credentials:** Authentication data (passwords, API keys) embedded directly in source code.
• **Command Injection:** A vulnerability that allows attackers to execute arbitrary commands on the host operating system.

Note: This report was automatically generated by the GitHubScan security tool. While comprehensive, it's recommended to have security experts review critical findings.