ZAP Scan report of localhost

Wes Ring

Sun, 11 Oct 2020 22:53:11

Contents

1 Findings for wesring.com **3**
1.1 High . . . 3
1.1.1 Remote OS Command Injection . . . 3
1.1.2 Path Traversal . . . 4
1.1.3 SQL Injection . . . 5
1.2 Medium . . . 6
1.2.1 X-Frame-Options Header Not Set . . . 6
1.2.2 Application Error Disclosure . . . 8
1.2.3 Directory Browsing . . . 9
1.2.4 Directory Browsing . . . 10
1.2.5 CSP Scanner: Wildcard Directive . . . 11
1.3 Low . . . 11
1.3.1 X-Content-Type-Options Header Missing . . . 11
1.3.2 Absence of Anti-CSRF Tokens . . . 13
1.3.3 Cross-Domain JavaScript Source File Inclusion . . . 15
1.3.4 Private IP Disclosure . . . 15
1.3.5 Application Error Disclosure . . . 16
1.3.6 Cookie No HttpOnly Flag . . . 16
1.3.7 Cookie Without SameSite Attribute . . . 17
1.3.8 Information Disclosure - Debug Error Messages . . . 17
1.4 Informational . . . 18
1.4.1 Timestamp Disclosure - Unix . . . 18
1.4.2 Information Disclosure - Suspicious Comments . . . 19
1.4.3 Information Disclosure - Sensitive Information in URL . . . 19
1.4.4 Information Disclosure - Suspicious Comments . . . 20
1.4.5 Loosely Scoped Cookie . . . 20

Abstract

This is a scan of localhost . This is blah blah blah boiler plate stuff.

1 Findings for wesring.com

Scan of localhost was done over port 80 and with SSL false.

1.1 High

1.1.1 Remote OS Command Injection

Confidence : 2

Description : Attack technique used for unauthorized execution of operating system commands. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs.

Solution : If at all possible, use library calls rather than external processes to recreate the desired functionality.Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the command locally in the session's state instead of sending it out to the client in a hidden form field.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, consider using the ESAPI Encoding control or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.If you need to use dynamically-generated query strings or commands in spite of the risk, properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict whitelist (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection.If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."When constructing OS command strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components.Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Instances

0	uri	http://localhost/vulnerabilities/exec/
	method	POST
	param	ip
	attack	ZAP&sleep 15&

Reference :

- `http://cwe.mitre.org/data/definitions/78.html`
- `https://owasp.org/www-community/attacks/Command{_}Injection`

1.1.2 Path Traversal

Confidence : 2

Description : The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\̈) on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character.Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (".") to reveal the listing of the current working directory, or "%00" NULL characters in order to bypass rudimentary file extension checks.

Solution : Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses, and exclude directory separators such as "/". Use a whitelist of allowable file extensions.Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links.Run your code using the lowest privileges that are required to accomplish the necessary tasks. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Instances

0	uri	http://localhost/vulnerabilities/fi/?page=%2Fetc%2Fpasswd
	method	GET
	param	page
	attack	/etc/passwd
	evidence	root:x:0:0

Reference :

- `http://projects.webappsec.org/Path-Traversal`
- `http://cwe.mitre.org/data/definitions/22.html`

1.1.3 SQL Injection

Confidence : 2

Description : SQL injection may be possible.

Solution : 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.

Instances

0	uri	http://localhost/vulnerabilities/upload/
	method	POST
	param	MAX_FILE_SIZE
	attack	100000 OR 1=1
1	uri	http://localhost/vulnerabilities/captcha/
	method	POST
	param	step
	attack	1 OR 1=1

Reference :

- `https://cheatsheetseries.owasp.org/cheatsheets/SQL{_}Injection{_}Prevention{_}Cheat{_}Sheet.html`

1.2 Medium

1.2.1 X-Frame-Options Header Not Set

Confidence : 2

Description : X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.

Solution : 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).

Instances

0	uri	http://localhost/dvwa/?C=S;O=D
	method	GET
	param	X-Frame-Options
1	uri	http://localhost/dvwa/js/
	method	GET
	param	X-Frame-Options
2	uri	http://localhost/dvwa/?C=D;O=A
	method	GET
	param	X-Frame-Options

	uri	http://localhost/dvwa/images/?C=N;O=A
3	method	GET
	param	X-Frame-Options
	uri	http://localhost/vulnerabilities/brute/
4	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/includes/DBMS/?C=D;O=A
5	method	GET
	param	X-Frame-Options
	uri	http://localhost/vulnerabilities/xss_d/?default
6	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/includes/DBMS/?C=S;O=A
7	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/?C=D;O=D
8	method	GET
	param	X-Frame-Options
	uri	http://localhost/about.php
9	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/includes/?C=S;O=A
10	method	GET
	param	X-Frame-Options
	uri	http://localhost/vulnerabilities/javascript/
11	method	POST
	param	X-Frame-Options
	uri	http://localhost/vulnerabilities/weak_id/
12	method	GET
	param	X-Frame-Options
	uri	http://localhost/instructions.php
13	method	GET
	param	X-Frame-Options
	uri	http://localhost/instructions.php?doc=PHPIDS-license
14	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/images/?C=N;O=D
15	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/images/
16	method	GET
	param	X-Frame-Options
	uri	http://localhost/dvwa/includes/DBMS/?C=D;O=D
17	method	GET
	param	X-Frame-Options
	uri	http://localhost/vulnerabilities/
18	method	GET
	param	X-Frame-Options

19	uri	http://localhost/dvwa/includes/?C=M;O=A
	method	GET
	param	X-Frame-Options

Reference :

- `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options`

1.2.2 Application Error Disclosure

Confidence : 2

Description : 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.

Solution : Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

Instances

0	uri	http://localhost/dvwa/images/?C=M;O=D
	method	GET
	evidence	Parent Directory
1	uri	http://localhost/dvwa/css/?C=M;O=A
	method	GET
	evidence	Parent Directory
2	uri	http://localhost/dvwa/includes/?C=D;O=D
	method	GET
	evidence	Parent Directory
3	uri	http://localhost/dvwa/images/?C=M;O=A
	method	GET
	evidence	Parent Directory
4	uri	http://localhost/dvwa/css/?C=M;O=D
	method	GET
	evidence	Parent Directory
5	uri	http://localhost/dvwa/includes/?C=D;O=A
	method	GET
	evidence	Parent Directory
6	uri	http://localhost/dvwa/includes/DBMS/?C=D;O=D
	method	GET
	evidence	Parent Directory
7	uri	http://localhost/dvwa/includes/DBMS/?C=D;O=A
	method	GET
	evidence	Parent Directory
8	uri	http://localhost/dvwa/includes/?C=M;O=D
	method	GET
	evidence	Parent Directory
9	uri	http://localhost/dvwa/js/?C=N;O=A
	method	GET
	evidence	Parent Directory

10	uri	http://localhost/dvwa/css/?C=D;O=A
	method	GET
	evidence	Parent Directory
11	uri	http://localhost/dvwa/js/?C=N;O=D
	method	GET
	evidence	Parent Directory
12	uri	http://localhost/vulnerabilities/?C=M;O=A
	method	GET
	evidence	Parent Directory
13	uri	http://localhost/dvwa/includes/DBMS/?C=M;O=D
	method	GET
	evidence	Parent Directory
14	uri	http://localhost/dvwa/css/?C=N;O=A
	method	GET
	evidence	Parent Directory
15	uri	http://localhost/dvwa/?C=D;O=D
	method	GET
	evidence	Parent Directory
16	uri	http://localhost/dvwa/includes/?C=N;O=D
	method	GET
	evidence	Parent Directory
17	uri	http://localhost/dvwa/images/?C=S;O=A
	method	GET
	evidence	Parent Directory
18	uri	http://localhost/dvwa/css/?C=N;O=D
	method	GET
	evidence	Parent Directory
19	uri	http://localhost/dvwa/includes/?C=N;O=A
	method	GET
	evidence	Parent Directory

Reference :

1.2.3 Directory Browsing

Confidence : 2

Description : It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files, etc. which can be accessed to read sensitive information.

Solution : Disable directory browsing. If this is required, make sure the listed files does not induce risks.

Instances

0	uri	http://localhost/dvwa/includes/
	method	GET
	attack	Parent Directory
1	uri	http://localhost/dvwa/images/
	method	GET
	attack	Parent Directory

	uri	http://localhost/dvwa/js/
2	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/includes/DBMS/
3	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/css/
4	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/
5	method	GET
	attack	Parent Directory

Reference :

- `http://httpd.apache.org/docs/mod/core.html#options`
- `http://alamo.satlug.org/pipermail/satlug/2002-February/000053.html`

1.2.4 Directory Browsing

Confidence : 1

Description : It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files, etc. which can be accessed to read sensitive information.

Solution : Disable directory browsing. If this is required, make sure the listed files does not induce risks.

Instances

	uri	http://localhost/dvwa/css/
0	method	GET
	attack	Parent Directory
	uri	http://localhost/docs/
1	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/
2	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/includes/DBMS/
3	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/js/
4	method	GET
	attack	Parent Directory
	uri	http://localhost/vulnerabilities/
5	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/images/
6	method	GET
	attack	Parent Directory
	uri	http://localhost/dvwa/includes/
7	method	GET
	attack	Parent Directory

Reference :

- `http://httpd.apache.org/docs/mod/core.html#options`
- `http://alamo.satlug.org/pipermail/satlug/2002-February/000053.html`

1.2.5 CSP Scanner: Wildcard Directive

Confidence : 2

Description : The following directives either allow wildcard sources (or ancestors), are not defined, or are overly broadly defined: style-src, style-src-elem, style-src-attr, img-src, connect-src, frame-src, frame-ancestors, font-src, media-src, object-src, manifest-src, prefetch-src

Solution : Ensure that your web server, application server, load balancer, etc. is properly configured to set the Content-Security-Policy header.

Instances

0	uri	http://localhost/vulnerabilities/csp/
	method	POST
	param	Content-Security-Policy
	evidence	script-src 'self' https://pastebin.com example.com code.jquery.com https://ssl.google-analytics.com ;
1	uri	http://localhost/vulnerabilities/csp/
	method	GET
	param	Content-Security-Policy
	evidence	script-src 'self' https://pastebin.com example.com code.jquery.com https://ssl.google-analytics.com ;

Reference :

- `http://www.w3.org/TR/CSP2/`
- `http://www.w3.org/TR/CSP/`
- `http://caniuse.com/#search=content+security+policy`
- `http://content-security-policy.com/`
- `https://github.com/shapesecurity/salvation`

1.3 Low

1.3.1 X-Content-Type-Options Header Missing

Confidence : 2

Description : 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.

Solution : 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.

Instances

	uri	http://localhost/dvwa/images/spanner.png
0	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/vulnerabilities/xss_d/?default
1	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/images/?C=N;O=A
2	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/includes/DBMS/?C=S;O=A
3	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/?C=D;O=A
4	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/?C=S;O=A
5	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/includes/dvwaPage.inc.php
6	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/includes/DBMS/?C=S;O=D
7	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/js/
8	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/instructions.php
9	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/css/help.css
10	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/includes/DBMS/?C=D;O=A
11	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/css/?C=N;O=A
12	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/about.php
13	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/css/main.css
14	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/vulnerabilities/sqli/
15	method	GET
	param	X-Content-Type-Options

	uri	http://localhost/dvwa/css/?C=N;O=D
16	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/?C=S;O=D
17	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/includes/?C=S;O=D
18	method	GET
	param	X-Content-Type-Options
	uri	http://localhost/dvwa/includes/?C=D;O=A
19	method	GET
	param	X-Content-Type-Options

Reference :

– `http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx`

– `https://owasp.org/www-community/Security{_}Headers`

1.3.2 Absence of Anti-CSRF Tokens

Confidence : 2

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.

Solution : Phase: Architecture and DesignUse 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: ImplementationEnsure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.Phase: Architecture and DesignGenerate 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: ImplementationCheck 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.

Instances

	uri	http://localhost/vulnerabilities/upload/
0	method	GET
	evidence	<form enctype=”multipart/form-data” action=”#” method=”POST”>
	uri	http://localhost/vulnerabilities/sqli/?Submit=Submit&id=ZAP
1	method	GET
	evidence	<form action=”#” method=”GET”>
	uri	http://localhost/vulnerabilities/weak_id/
2	method	GET
	evidence	<form method=”post”>
	uri	http://localhost/vulnerabilities/javascript/
3	method	GET
	evidence	<form name=”low_js” method=”post”>
	uri	http://localhost/vulnerabilities/brute/?Login=Login&password=ZAP&username=ZAP
4	method	GET
	evidence	<form action=”#” method=”GET”>
	uri	http://localhost/login.php
5	method	GET
	evidence	<form action=”login.php” method=”post”>
	uri	http://localhost/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&password_new=ZAP
6	method	GET
	evidence	<form action=”#” method=”GET”>
	uri	http://localhost/vulnerabilities/exec/
7	method	POST
	evidence	<form name=”ping” action=”#” method=”post”>
	uri	http://localhost/vulnerabilities/sqli_blind/?Submit=Submit&id=ZAP
8	method	GET
	evidence	<form action=”#” method=”GET”>
	uri	http://localhost/vulnerabilities/weak_id/
9	method	POST
	evidence	<form method=”post”>
	uri	http://localhost/vulnerabilities/xss_s/
10	method	POST
	evidence	<form method=”post” name=”guestform” ”>
	uri	http://localhost/vulnerabilities/view_source.php
11	method	GET
	evidence	<form>
	uri	http://localhost/vulnerabilities/xss_r/
12	method	GET
	evidence	<form name=”XSS” action=”#” method=”GET”>
	uri	http://localhost/vulnerabilities/sqli_blind/
13	method	GET
	evidence	<form action=”#” method=”GET”>
	uri	http://localhost/vulnerabilities/xss_r/?name=ZAP
14	method	GET
	evidence	<form name=”XSS” action=”#” method=”GET”>
	uri	http://localhost/vulnerabilities/exec/
15	method	GET
	evidence	<form name=”ping” action=”#” method=”post”>

16	uri	http://localhost/vulnerabilities/sqli/
	method	GET
	evidence	<form action="#" method="GET">
17	uri	http://localhost/vulnerabilities/xss_s/
	method	GET
	evidence	<form method="post" name="guestform" ">
18	uri	http://localhost/vulnerabilities/captcha/
	method	GET
	evidence	<form action="#" method="POST" style="display:none;">
19	uri	http://localhost/setup.php
	method	GET
	evidence	<form action="#" method="post">

Reference :

- `http://projects.webappsec.org/Cross-Site-Request-Forgery`
- `http://cwe.mitre.org/data/definitions/352.html`

1.3.3 Cross-Domain JavaScript Source File Inclusion

Confidence : 2

Description : The page includes one or more script files from a third-party domain.

Solution : Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.

Instances

0	uri	http://localhost/vulnerabilities/captcha/
	method	GET
	param	https://www.google.com/recaptcha/api.js
	evidence	<script src='https://www.google.com/recaptcha/api.js'></script>
1	uri	http://localhost/vulnerabilities/captcha/
	method	POST
	param	https://www.google.com/recaptcha/api.js
	evidence	<script src='https://www.google.com/recaptcha/api.js'></script>

Reference :

1.3.4 Private IP Disclosure

Confidence : 2

Description : 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 found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.

Solution : 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.

Instances

0	uri	http://localhost/vulnerabilities/fi/?page=file1.php
	method	GET
	evidence	172.17.0.1

1	uri	http://localhost/vulnerabilities/fi/?page=file3.php
	method	GET
	evidence	172.17.0.1
2	uri	http://localhost/ids_log.php
	method	GET
	evidence	172.17.0.2
3	uri	http://localhost/phpinfo.php
	method	GET
	evidence	172.17.0.2:80

Reference :

– `https://tools.ietf.org/html/rfc1918`

1.3.5 Application Error Disclosure

Confidence : 2

Description : 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.

Solution : Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

Instances

0	uri	http://localhost/dvwa/includes/DBMS/MySQL.php
	method	GET
	evidence	HTTP/1.0 500 Internal Server Error
1	uri	http://localhost/dvwa/includes/DBMS/PGSQL.php
	method	GET
	evidence	HTTP/1.0 500 Internal Server Error

Reference :

1.3.6 Cookie No HttpOnly Flag

Confidence : 2

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.

Solution : Ensure that the HttpOnly flag is set for all cookies.

Instances

0	uri	http://localhost/
	method	GET
	param	PHPSESSID
	evidence	Set-Cookie: PHPSESSID
1	uri	http://localhost/vulnerabilities/weak_id/
	method	POST
	param	dvwaSession
	evidence	Set-Cookie: dvwaSession

2	uri	http://localhost/
	method	GET
	param	security
	evidence	Set-Cookie: security

Reference :

– `https://owasp.org/www-community/HttpOnly`

1.3.7 Cookie Without SameSite Attribute

Confidence : 2

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.

Solution : Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.

Instances

0	uri	http://localhost/
	method	GET
	param	security
	evidence	Set-Cookie: security
1	uri	http://localhost/
	method	GET
	param	PHPSESSID
	evidence	Set-Cookie: PHPSESSID
2	uri	http://localhost/vulnerabilities/weak_id/
	method	POST
	param	dvwaSession
	evidence	Set-Cookie: dvwaSession

Reference :

– `https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site`

1.3.8 Information Disclosure - Debug Error Messages

Confidence : 2

Description : The response appeared to contain common error messages returned by platforms such as ASP.NET, and Web-servers such as IIS and Apache. You can configure the list of common debug messages.

Solution : Disable debugging messages before pushing to production.

Instances

0	uri	http://localhost/instructions.php
	method	GET
	evidence	PHP warning
1	uri	http://localhost/instructions.php?doc=readme
	method	GET
	evidence	PHP warning

Reference :

1.4 Informational

1.4.1 Timestamp Disclosure - Unix

Confidence : 1

Description : A timestamp was disclosed by the application/web server - Unix

Solution : Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.

Instances

0	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000071009
1	uri	http://localhost/vulnerabilities/javascript/
	method	POST
	evidence	606105819
2	uri	http://localhost/vulnerabilities/javascript/
	method	GET
	evidence	1894986606
3	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000339686
4	uri	http://localhost/vulnerabilities/javascript/
	method	GET
	evidence	1094730640
5	uri	http://localhost/vulnerabilities/javascript/
	method	POST
	evidence	405537848
6	uri	http://localhost/vulnerabilities/javascript/
	method	POST
	evidence	187363961
7	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000341194
8	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000345972
9	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000339382
10	uri	http://localhost/phpinfo.php
	method	GET
	evidence	20150407
11	uri	http://localhost/vulnerabilities/javascript/
	method	POST
	evidence	681279174
12	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000042762

13	uri	http://localhost/vulnerabilities/javascript/
	method	POST
	evidence	155497632
14	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000341610
15	uri	http://localhost/vulnerabilities/javascript/
	method	GET
	evidence	1444681467
16	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000001052
17	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000345158
18	uri	http://localhost/docs/DVWA_v1.3.pdf
	method	GET
	evidence	0000343297
19	uri	http://localhost/vulnerabilities/javascript/
	method	POST
	evidence	1839030562

Reference :

– `http://projects.webappsec.org/w/page/13246936/Information%20Leakage`

1.4.2 Information Disclosure - Suspicious Comments

Confidence : 2

Description : The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.

Solution : Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.

Instances

0	uri	http://localhost/setup.php
	method	GET

Reference :

1.4.3 Information Disclosure - Sensitive Information in URL

Confidence : 2

Description : The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment.

Solution : Do not pass sensitive information in URIs.

Instances

0	uri	http://localhost/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&password_new=ZAP
	method	GET
	param	password_new
	evidence	password_new
1	uri	http://localhost/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&password_new=ZAP
	method	GET
	param	password_conf
	evidence	password_conf
2	uri	http://localhost/vulnerabilities/brute/?Login=Login&password=ZAP&username=ZAP
	method	GET
	param	username
	evidence	username
3	uri	http://localhost/vulnerabilities/brute/?Login=Login&password=ZAP&username=ZAP
	method	GET
	param	password
	evidence	password

Reference :

1.4.4 Information Disclosure - Suspicious Comments

Confidence : 1

Description : The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.

Solution : Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.

Instances

0	uri	http://localhost/vulnerabilities/javascript/
	method	POST
1	uri	http://localhost/vulnerabilities/javascript/
	method	GET

Reference :

1.4.5 Loosely Scoped Cookie

Confidence : 1

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.

Solution : Always scope cookies to a FQDN (Fully Qualified Domain Name).

Instances

0	uri	http://localhost/
	method	GET

1	uri	http://localhost/
	method	GET
2	uri	http://localhost/vulnerabilities/weak_id/
	method	POST
3	uri	http://localhost/
	method	GET

Reference :

- `https://tools.ietf.org/html/rfc6265#section-4.1`
- `https://owasp.org/www-project-web-security-testing-guide/v41/4-Web{_}Application{_}Security{_}`
  `06-Session{_}Management{_}Testing/02-Testing{_}for{_}Cookies{_}Attributes.html`
- `http://code.google.com/p/browsersec/wiki/Part2#Same-origin{_}policy{_}for{_}cookies`