ZAP Scanning Report

Site: http://localhost:8080

Generated on Sun, 25 Dec 2022 22:09:21

Summary of Alerts

Risk Level	Number of Alerts
High	7
Medium	3
Low	4
Informational	6

Alerts

Name	Risk Level	Number of Instances
Cross Site Scripting (DOM Based)	High	3
Cross Site Scripting (Persistent)	High	4
Cross Site Scripting (Reflected)	High	3
Path Traversal	High	1
SQL Injection	High	2
SQL Injection - PostgreSQL	High	4
SQL Injection - PostgreSQL - Time Based	High	3
Absence of Anti-CSRF Tokens	Medium	15
Content Security Policy (CSP) Header Not Set	Medium	16
Missing Anti-clickjacking Header	Medium	16
Cookie No HttpOnly Flag	Low	3
Cookie without SameSite Attribute	Low	4
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	19
X-Content-Type-Options Header Missing	Low	16
Cookie Poisoning	Informational	20
Information Disclosure - Sensitive Information in URL	Informational	4
Information Disclosure - Suspicious Comments	Informational	1
Loosely Scoped Cookie	Informational	5
Modern Web Application	Informational	3
User Controllable HTML Element Attribute (Potential XSS)	Informational	20

Alert Detail

High	Cross Site Scripting (DOM Based)

Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML /JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://localhost:8080/part1_vuln#jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e
Method	GET
Attack	#jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e
Evidence	
URL	http://localhost:8080/part3_vuln#jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e
Method	GET
Attack	#jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e
Evidence	
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123#jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e
Method	GET
Attack	#jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert(5397))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert(5397)//>\x3e
Evidence	
Instances	3
	Phase: Architecture and Design

Solution	Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. 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. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list 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 deny list). However, deny lists 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." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html

CWE Id	79
WASC Id	8
Plugin Id	40026

High	Cross Site Scripting (Persistent)
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML /JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	<script>alert(1);</script>
Evidence	
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	<script>alert(1);</script>
Evidence	
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	<script>alert(1);</script>
Evidence	
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	<script>alert(1);</script>

Evidence	
Instances	4
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. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. 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. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list 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 deny list). However, deny lists 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."

	Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Plugin Id	40014

High	**Cross Site Scripting (Reflected)**
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code i standard web browser client, or a browser object embedded in a software product such as the b code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java When an attacker gets a user's browser to execute his/her code, the code will run within the sec privilege, the code has the ability to read, modify and transmit any sensitive data accessible by t hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulen Scripting attacks essentially compromise the trust relationship between a user and the web site. from the file system may execute code under the local machine zone allowing for system compr There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted l containing a web form, which when posted to the vulnerable site, will mount the attack. Using a resource only accepts HTTP POST requests. In such a case, the form can be submitted automa Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get ec execute. Another technique to send almost arbitrary requests (GET and POST) is by using an e Persistent attacks occur when the malicious code is submitted to a web site where it's stored for include message board posts, web mail messages, and web chat software. The unsuspecting u attacker site or a malicious link sent via email), just simply view the web page containing the coc
URL	http://localhost:8080/part2_vuln.html?v_text=%3C%2Fli%3E%3CscrIpt%3Ealert%281%29%3B%
Method	GET
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 3Cimg+src%3Dx+onerror%3Dprompt%28%29% 3E&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end
Method	GET
Attack	<img src=x onerror=prompt()>
Evidence	<img src=x onerror=prompt()>
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_ 281%29%3B%3C%2FscRipt%3E%3C%21--&v_sp_m=1&v_sp_s=0
Method	GET
Attack	--><scrIpt>alert(1);</scRipt><!--
Evidence	--><scrIpt>alert(1);</scRipt><!--
Instances	3
	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constru Examples of libraries and frameworks that make it easier to generate properly encoded output i module, and Apache Wicket.

Solution	Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. T different components, or when generating outputs that can contain multiple encodings at the sa expected communication protocols and data representations to determine the required encodin For any data that will be output to another web page, especially any data that was received fro alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escapin Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are dup can bypass the client-side checks by modifying values after the checks have been performed, o Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data quoting, encoding, and validation automatically, instead of relying on the developer to provide th Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859- may choose a different encoding by guessing which encoding is actually being used by the web as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations relat To help mitigate XSS attacks against the user's session cookie, set the session cookie to be Htt more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's sess that use document.cookie. This is not a complete solution, since HttpOnly is not supported by al powerful browser technologies provide read access to HTTP headers, including the Set-Cookie Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an specifications. Reject any input that does not strictly conform to specifications, or transform it int malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be usefu malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, ty inputs, syntax, consistency across related fields, and conformance to business rules. As an exa because it only contains alphanumeric characters, but it is not valid if you are expecting colors s Ensure that you perform input validation at well-defined interfaces within the application. This wi moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Plugin Id	40012

High	**Path Traversal**
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.
URL	http://localhost:8080/register_vuln.html?v_username=register_vuln.html&v_password=Password_123
Method	GET
Attack	register_vuln.html
Evidence	
Instances	1
Solution	Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list 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 deny list). However, deny lists 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 allow lists 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 an allow list 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 allow list 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.
Reference	http://projects.webappsec.org/Path-Traversal http://cwe.mitre.org/data/definitions/22.html
CWE Id	22
WASC Id	33
Plugin Id	6

High	**SQL Injection**
Description	SQL injection may be possible
URL	http://localhost:8080/part1_vuln.html?v_username=admin%27+AND+%271%27%3D%271%27+--+&v_password=password&v_remember=on
Method	GET
Attack	admin' AND '1'='1' --
Evidence	
URL	http://localhost:8080/register_vuln.html?v_username=user123%27+AND+%271%27%3D%271%27+--+&v_password=Password_123
Method	GET
Attack	user123' AND '1'='1' --
Evidence	
Instances	2
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 an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input. Apply the privilege 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.
	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.

Reference	html
CWE Id	89
WASC Id	19
Plugin Id	40018

High	SQL Injection - PostgreSQL
Description	SQL injection may be possible
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	asd') UNION ALL select NULL --
Evidence	each UNION query must have the same number of columns
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	') UNION ALL select NULL --
Evidence	each UNION query must have the same number of columns
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	') UNION ALL select NULL --
Evidence	each UNION query must have the same number of columns
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	') UNION ALL select NULL --
Evidence	each UNION query must have the same number of columns
Instances	4
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 pa If the application uses ASP, use ADO Command Objects with strong type checking and parame If database Stored Procedures can be used, use them. Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate 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 privilege 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 inje 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

High	**SQL Injection - PostgreSQL - Time Based**
Description	SQL injection may be possible
URL	http://localhost:8080/part3_vuln.html?v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	field: [v_pricemax], value [case when cast(pg_sleep(15) as varchar) > '' then 0 else 1 end]
Evidence	
URL	http://localhost:8080/part3_vuln.html?v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	field: [v_pricemin], value [case when cast(pg_sleep(15) as varchar) > '' then 0 else 1 end]
Evidence	
URL	http://localhost:8080/part3_vuln.html?v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	field: [v_sp_c], value [case when cast(pg_sleep(15) as varchar) > '' then 0 else 1 end]
Evidence	
Instances	3
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 pa If the application uses ASP, use ADO Command Objects with strong type checking and parame If database Stored Procedures can be used, use them. Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediat 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 privilege 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 inje 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	40022

Medium	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 t perform an action as the victim. The underlying cause is application functionality using predictab is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XS CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also 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 privile information by gaining access to the response. The risk of information disclosure is dramatically XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of th
URL	http://localhost:8080/part1_correct
Method	GET
Attack	
Evidence	<form action="/part1_correct.html" method="post">
URL	http://localhost:8080/part1_vuln
Method	GET
Attack	
Evidence	<form action="/part1_vuln.html">
URL	http://localhost:8080/part2_correct
Method	GET
Attack	
Evidence	<form action="/part2_correct.html">
URL	http://localhost:8080/part2_correct.html?c_text=+asd
Method	GET
Attack	
Evidence	<form action="/part2_correct.html">
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	
Evidence	<form action="/part2_vuln.html">
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	
Evidence	<form action="/part2_vuln.html">
URL	http://localhost:8080/part3_correct
Method	GET
Attack	
Evidence	<form action="/part3_correct.html">

URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	<form action="/part3_correct.html">
URL	http://localhost:8080/part3_vuln
Method	GET
Attack	
Evidence	<form action="/part3_vuln.html">
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	<form action="/part3_vuln.html">
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	<form>
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	<form method="POST">
URL	http://localhost:8080/register_correct
Method	GET
Attack	
Evidence	<form action="/register_correct.html" method="post">
URL	http://localhost:8080/register_vuln
Method	GET
Attack	
Evidence	<form action="/register_vuln.html">
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	<form action="/register_vuln.html">
Instances	15
	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constru 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

Solution	Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upo (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send 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 co 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

Medium	**Content Security Policy (CSP) Header Not Set**
Description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate cer data injection attacks. These attacks are used for everything from data theft to site defacement headers that allow website owners to declare approved sources of content that browsers should CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, aud
URL	http://localhost:8080/
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remembe
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_correct
Method	GET

Attack	
Evidence	
URL	http://localhost:8080/part2_correct.html?c_text=+asd
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_vuln

Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	
Instances	16
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Cont "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, "X-Content-Security-Policy CSP" for Chrome 14+ and Safari 6+.
Reference	https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Poli https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html http://www.w3.org/TR/CSP/ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html http://www.html5rocks.com/en/tutorials/security/content-security-policy/ http://caniuse.com/#feat=contentsecuritypolicy http://content-security-policy.com/
CWE Id	693
WASC Id	15
Plugin Id	10038

Medium	**Missing Anti-clickjacking Header**
Description	The response does not include either Content-Security-Policy with 'frame-ancestors' directive or
URL	http://localhost:8080/
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remembe
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_correct.html?c_text=+asd

Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_vuln
Method	GET
Attack	
Evidence	

URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	
Instances	16
Solution	Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP header site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET expect the page to be framed, you should use DENY. Alternatively consider implementing Cont
Reference	https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
CWE Id	1021
WASC Id	15
Plugin Id	10020

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/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	Set-Cookie: password
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	Set-Cookie: username
URL	http://localhost:8080/part1_correct.html
Method	POST
Attack	
Evidence	Set-Cookie: d845663eaea45f7b797ed78273d8acf498ed4533e0d91ac9f0c3c51d757531da
Instances	3
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/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET

Attack	
Evidence	Set-Cookie: password
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	Set-Cookie: username
URL	http://localhost:8080/part1_correct.html
Method	POST
Attack	
Evidence	Set-Cookie: d845663eaea45f7b797ed78273d8acf498ed4533e0d91ac9f0c3c51d757531da
URL	http://localhost:8080/part1_correct.html
Method	POST
Attack	
Evidence	Set-Cookie: session
Instances	4
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	**Server Leaks Version Information via "Server" HTTP Response Header Field**
Description	The web/application server is leaking version information via the "Server" HTTP response head other vulnerabilities your web/application server is subject to.
URL	http://localhost:8080/
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/logout
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part1_correct
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part1_vuln
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remembe
Method	GET

Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part2_correct
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part2_correct.html?c_text=+asd
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part3_correct
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part3_vuln
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1

URL	http://localhost:8080/register_correct
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/register_vuln
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/part1_correct.html
Method	POST
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
URL	http://localhost:8080/register_correct.html
Method	POST
Attack	
Evidence	Werkzeug/2.2.2 Python/3.11.1
Instances	19
Solution	Ensure that your web server, application server, load balancer, etc. is configured to suppress th
Reference	http://httpd.apache.org/docs/current/mod/core.html#servertokens http://msdn.microsoft.com/en-us/library/ff648552.aspx#ht_urlscan_007 http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.a http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html
CWE Id	200
WASC Id	13
Plugin Id	10036

Low	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows old sniffing on the response body, potentially causing the response body to be interpreted and displ Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is s
URL	http://localhost:8080/
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln
Method	GET

Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remembe
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_correct.html?c_text=+asd
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part2_vuln.html?v_text=++++++++++++++++++++++++++++++++asd
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2

Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_correct
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_vuln
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	
Instances	16
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it s pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that the web application/web server to not perform MIME-sniffing.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://owasp.org/www-community/Security_Headers
CWE Id	693
WASC Id	15
Plugin Id	10021

Informational	**Cookie Poisoning**
Description	This check looks at user-supplied input in query string parameters and POST data to identify where cookie parameters might be controlled. This is called a cookie poisoning attack, and becomes exploitable when an attacker can manipulate the cookie in various ways. In some cases this will not be exploitable, however, allowing URL parameters to set cookie values is generally considered a bug.
URL	http://localhost:8080/part1_vuln.html? v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html? v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html? v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	

Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	

Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
Instances	20
Solution	Do not allow user input to control cookie names and values. If some query string parameters must be set in cookie values, be sure to filter out semicolon's that can serve as name/value pair delimiters.
Reference	http://websecuritytool.codeplex.com/wikipage?title=Checks#user-controlled-cookie

CWE Id	20
WASC Id	20
Plugin Id	10029

Informational	Information Disclosure - Sensitive Information in URL
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.
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	v_password
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	v_username
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	v_password
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	v_username
Instances	4
Solution	Do not pass sensitive information in URIs.
Reference	
CWE Id	200
WASC Id	13
Plugin Id	10024

Informational	Information Disclosure - Suspicious Comments
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.
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	select
Instances	1
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Reference	
CWE Id	200
WASC Id	13

Plugin Id	10027

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/logout
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/logout
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_vuln.html?v_username=admin&v_password=password&v_remember=on
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/register_vuln.html?v_username=user123&v_password=Password_123
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part1_correct.html
Method	POST
Attack	
Evidence	
Instances	5
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

Informational	Modern Web Application
Description	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.
URL	http://localhost:8080/part2_vuln
Method	GET

Attack	
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://localhost:8080/part2_vuln.html?v_text=+++++++++++++++++++++++++++++++asd
Method	GET
Attack	
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://localhost:8080/register/2fa/user321/725ed279-06d0-4947-84f2-db1c7df93ca2
Method	GET
Attack	
Evidence	<a>- Download
Instances	3
Solution	This is an informational alert and so no changes are required.
Reference	
CWE Id	
WASC Id	
Plugin Id	[10109](10109)

Informational	**User Controllable HTML Element Attribute (Potential XSS)**
Description	This check looks at user-supplied input in query string parameters and POST data to identify wh provides hot-spot detection for XSS (cross-site scripting) that will require further review by a sec
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie

	1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_correct.html? c_name=&c_author=&c_category=&c_pricemin=&c_pricemax=&c_search_input=&c_search_fie 1&c_sp_start_month=0&c_sp_start_day=0&c_sp_start_year=&c_sp_end_month=0&c_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie

	1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
URL	http://localhost:8080/part3_vuln.html? v_name=&v_author=&v_category=&v_pricemin=&v_pricemax=&v_search_input=&v_search_fie 1&v_sp_start_month=0&v_sp_start_day=0&v_sp_start_year=&v_sp_end_month=0&v_sp_end_
Method	GET
Attack	
Evidence	
Instances	20

Solution	Validate all input and sanitize output it before writing to any HTML attributes.
Reference	http://websecuritytool.codeplex.com/wikipage?title=Checks#user-controlled-html-attribute
CWE Id	20
WASC Id	20
Plugin Id	10031