ZAP Scanning Report

Sites: https://192.168.0.16:8061 https://192.168.0.16:4300

Generated on Sun, 6 Feb 2022 16:23:52

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	5
Informational	1

Alerts

Name	Risk Level	Number of Instances
Buffer Overflow	Medium	1
Cross-Domain Misconfiguration	Medium	8
Missing Anti-clickjacking Header	Medium	1
Cross Site Scripting Weakness (Reflected in JSON Response)	Low	1
Incomplete or No Cache-control Header Set	Low	1
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	8
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	8
Information Disclosure - Suspicious Comments	Informational	3

Alert Detail

Medium	Buffer Overflow
Description	Buffer overflow errors are characterized by the overwriting of memory spaces of the backgrounc
URL	https://192.168.0.16:8061/account/register
Method	POST
Attack	OgBTFDDHTVHiuhJPScEUtpMRsowJhvUedjwZVSBxlrXmpUHgFffXrqjNYiYtsGjrqOtGuhxQTxF
Evidence	Connection: close
Instances	1
Solution	Rewrite the background program using proper return length checking. This will require a recomp
Reference	https://owasp.org/www-community/attacks/Buffer_overflow_attack
CWE Id	120
WASC Id	7
Plugin Id	30001

Medium	Cross-Domain Misconfiguration
Description	Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server
URL	https://192.168.0.16:4300/
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/assets/logo.png
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/favicon.ico
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/main.js
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/polyfills.js
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/runtime.js
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/styles.css
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	https://192.168.0.16:4300/styles.js
Method	GET
Attack	
Evidence	Access-Control-Allow-Origin: *
Instances	8
Solution	Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.
Reference	https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy

CWE Id	264
WASC Id	14
Plugin Id	10098

Medium	**Missing Anti-clickjacking Header**
Description	The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
URL	https://192.168.0.16:4300/
Method	GET
Attack	
Evidence	
Instances	1
Solution	Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. 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. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.
Reference	https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
CWE Id	1021
WASC Id	15
Plugin Id	10020

Low	**Cross Site Scripting Weakness (Reflected in JSON Response)**
Description	A XSS attack was reflected in a JSON response, this might leave content consumers vulnerable to attack if they don't appropriately handle the data (response).
URL	https://192.168.0.16:8061/account/register
Method	POST
Attack	<img src=x onerror=prompt()>
Evidence	
Instances	1
	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.

Solution	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	40012

Low	**Incomplete or No Cache-control Header Set**
Description	The cache-control header has not been set properly or is missing, allowing the browser and proxies to cache content.
URL	https://192.168.0.16:4300/
Method	GET
Attack	
Evidence	
Instances	1
Solution	Whenever possible ensure the cache-control HTTP header is set with no-cache, no-store, must-revalidate.

Reference	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#web-content-caching https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control
CWE Id	525
WASC Id	13
Plugin Id	10015

Low	**Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)**
Description	The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.
URL	https://192.168.0.16:4300/
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/assets/logo.png
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/favicon.ico
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/main.js
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/polyfills.js
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/runtime.js
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/styles.css
Method	GET
Attack	
Evidence	X-Powered-By: Express
URL	https://192.168.0.16:4300/styles.js
Method	GET
Attack	
Evidence	X-Powered-By: Express

Instances	8
Solution	Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.
Reference	http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html
CWE Id	200
WASC Id	13
Plugin Id	10037

Low	**Timestamp Disclosure - Unix**
Description	A timestamp was disclosed by the application/web server - Unix
URL	https://192.168.0.16:4300/main.js
Method	GET
Attack	
Evidence	2147483647
URL	https://192.168.0.16:4300/polyfills.js
Method	GET
Attack	
Evidence	2147483647
URL	https://192.168.0.16:4300/styles.css
Method	GET
Attack	
Evidence	00000024
URL	https://192.168.0.16:4300/styles.css
Method	GET
Attack	
Evidence	00000061
URL	https://192.168.0.16:4300/styles.js
Method	GET
Attack	
Evidence	2147483647
Instances	5
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage
CWE Id	200
WASC Id	13
Plugin Id	10096

Low	**X-Content-Type-Options Header Missing**
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.
URL	https://192.168.0.16:4300/
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/assets/logo.png
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/favicon.ico
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/main.js
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/polyfills.js
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/runtime.js
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/styles.css
Method	GET
Attack	
Evidence	
URL	https://192.168.0.16:4300/styles.js
Method	GET
Attack	
Evidence	
Instances	8
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.
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	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	https://192.168.0.16:4300/main.js
Method	GET
Attack	
Evidence	query
URL	https://192.168.0.16:4300/polyfills.js
Method	GET
Attack	
Evidence	query
URL	https://192.168.0.16:4300/styles.js
Method	GET
Attack	
Evidence	query
Instances	3
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