Semgrep SAST Scan Report for Repository: Semgrep-Demo/juice-shop

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	29
Findings- SAST Medium Severity	75
Findings- SAST Low Severity	4

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/dataExport.ts#L61
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/dataExport.ts#L80
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/likeProductReviews.ts#L18
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly	high	open	main	routes/likeProductReviews.ts#L25

Finding Title	Finding Description & Remediation	severity	status	ref	location
	sanitize the data if you absolutely must pass request data into a mongo query.				
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/likeProductReviews.ts#L31
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/likeProductReviews.ts#L42
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/orderHistory.ts#L17
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/orderHistory.ts#L36

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/showProductReviews.ts#L34
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/trackOrder.ts#L17
express-mongo-nosqli	Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to properly sanitize the data if you absolutely must pass request data into a mongo query.	high	open	main	routes/updateProductReviews.ts#L18
tainted-sql-string	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational	high	open	main	data/static/codefixes/dbSchemaChallenge_1.ts#L5

Finding Title	Finding Description & Remediation	severity	status	ref	location
	mapper (ORM) such as Sequelize which will protect your queries.				
tainted-sql-string	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.	high	open	main	data/static/codefixes/dbSchemaChallenge_3.ts#L11
tainted-sql-string	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.	high	open	main	data/static/codefixes/unionSqlInjectionChallenge_1.ts#L6
tainted-sql-string	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of	high	open	main	data/static/codefixes/unionSqlInjectionChallenge_3.ts#L10

Finding Title	Finding Description & Remediation	severity	status	ref	location
	the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.				
[tainted-sql-string]()	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.	high	open	main	[routes/login.ts#L36]()
[tainted-sql-string]()	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.	high	open	main	[routes/search.ts#L23]()
	Detected a sequelize statement that is tainted by user-input. This could lead to	high	ignored	main	[data/static/codefixes/dbSchemaChallenge_1.ts#L5]()

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-sequelize-injection	SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.				
express-sequelize-injection	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.	high	open	main	data/static/codefixes/dbSchemaChallenge_3.ts#L11
express-sequelize-injection	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.	high	open	main	data/static/codefixes/unionSqlInjectionChallenge_1.ts#L6
express-sequelize-injection	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.	high	open	main	data/static/codefixes/unionSqlInjectionChallenge_3.ts#L10
express-sequelize-injection	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is	high	open	main	routes/login.ts#L36

Finding Title	Finding Description & Remediation	severity	status	ref	location
	recommended to use parameterized queries or prepared statements.				
express-sequelize-injection	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.	high	open	main	routes/search.ts#L23
detected-generic-secret	Generic Secret detected	high	open	main	data/static/users.yml#L150
insecure-document-method	User controlled data in methods like \`innerHTML\`, \`outerHTML\` or \`document.write\` is an anti-pattern that can lead to XSS vulnerabilities	high	open	main	frontend/src/hacking-instructor/index.ts#L107
crlf-injection-logs-deepsemgrep	When data from an untrusted source is put into a logger and not neutralized correctly, an attacker could forge log entries or include malicious content.	high	open	refs/pull/2/merge	src/assistant-fix-custom-message.java#L14
crlf-injection-logs-deepsemgrep-javaorg-copy	When data from an untrusted source is put into a logger and not neutralized correctly, an attacker could forge log entries or include malicious content. Please use the Jsoup.clean() function to sanitize data.	high	reviewing	refs/pull/2/merge	src/assistant-fix-custom-message.java#L14
tainted-sql-string	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL	high	open	refs/pull/1/merge	src/assistant-fix-sqli-sequelize.ts#L5

Finding Title	Finding Description & Remediation	severity	status	ref	location
	injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using an object-relational mapper (ORM) such as Sequelize which will protect your queries.				
express-sequelize-injection	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it is recommended to use parameterized queries or prepared statements.	high	open	refs/pull/1/merge	src/assistant-fix-sqli-sequelize.ts#L5

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/fileServer.ts#L33
express-fs-filename	The application builds a file path from potentially untrusted data, which can lead to a path traversal vulnerability. An attacker can manipulate the file path which the application uses to access files. If the application does not validate user input and sanitize file paths, sensitive files such as configuration or user data can be accessed, potentially creating or overwriting files. To prevent this vulnerability, validate and sanitize any input that is used to create references to file paths. Also, enforce strict file access controls. For example, choose privileges allowing public-facing applications to access only the required files.	medium	open	main	routes/profileImageFileUpload.ts#L28
express-fs-filename	The application builds a file path from potentially untrusted data, which can lead to a path traversal vulnerability. An attacker can manipulate the file path which the application uses to access	medium	open	main	routes/profileImageUrlUpload.ts#L31

Finding Title	Finding Description & Remediation	severity	status	ref	location
	files. If the application does not validate user input and sanitize file paths, sensitive files such as configuration or user data can be accessed, potentially creating or overwriting files. To prevent this vulnerability, validate and sanitize any input that is used to create references to file paths. Also, enforce strict file access controls. For example, choose privileges allowing public-facing applications to access only the required files.				
express-fs-filename	The application builds a file path from potentially untrusted data, which can lead to a path traversal vulnerability. An attacker can manipulate the file path which the application uses to access files. If the application does not validate user input and sanitize file paths, sensitive files such as configuration or user data can be accessed, potentially creating or overwriting files. To prevent this vulnerability, validate and sanitize any input that is used to create references to file paths. Also, enforce strict file access controls. For example, choose privileges allowing public-facing applications to access only the required files.	medium	open	main	routes/vulnCodeFixes.ts#L79
express-fs-filename	The application builds a file path from potentially untrusted data, which can lead to a path traversal vulnerability. An attacker can manipulate the file path	medium	open	main	routes/vulnCodeFixes.ts#L80

Finding Title	Finding Description & Remediation	severity	status	ref	location
	which the application uses to access files. If the application does not validate user input and sanitize file paths, sensitive files such as configuration or user data can be accessed, potentially creating or overwriting files. To prevent this vulnerability, validate and sanitize any input that is used to create references to file paths. Also, enforce strict file access controls. For example, choose privileges allowing public-facing applications to access only the required files.				
express-fs-filename	The application builds a file path from potentially untrusted data, which can lead to a path traversal vulnerability. An attacker can manipulate the file path which the application uses to access files. If the application does not validate user input and sanitize file paths, sensitive files such as configuration or user data can be accessed, potentially creating or overwriting files. To prevent this vulnerability, validate and sanitize any input that is used to create references to file paths. Also, enforce strict file access controls. For example, choose privileges allowing public-facing applications to access only the required files.	medium	open	main	routes/vulnCodeSnippet.ts#L93
express-fs-filename	The application builds a file path from potentially untrusted data, which can lead to a path traversal vulnerability. An	medium	open	main	routes/vulnCodeSnippet.ts#L94

Finding Title	Finding Description & Remediation	severity	status	ref	location
	attacker can manipulate the file path which the application uses to access files. If the application does not validate user input and sanitize file paths, sensitive files such as configuration or user data can be accessed, potentially creating or overwriting files. To prevent this vulnerability, validate and sanitize any input that is used to create references to file paths. Also, enforce strict file access controls. For example, choose privileges allowing public-facing applications to access only the required files.				
open-redirect-deepsemgrep	The application builds a URL using user-controlled input which can lead to an open redirect vulnerability. An attacker can manipulate the URL and redirect users to an arbitrary domain. Open redirect vulnerabilities can lead to issues such as Cross-site scripting (XSS) or redirecting to a malicious domain for activities such as phishing to capture users' credentials. To prevent this vulnerability perform strict input validation of the domain against an allowlist of approved domains. Notify a user in your application that they are leaving the website. Display a domain where they are redirected to the user. A user can then either accept or deny the redirect to an untrusted site.	medium	ignored	main	routes/redirect.ts#L19
		medium	open	main	routes/profileImageUrlUpload.ts#L23

Finding Title	Finding Description & Remediation	severity	status	ref	location
ssrf-deepsemgrep	Untrusted input might be used to build an HTTP request, which can lead to a Server-side request forgery (SSRF) vulnerability. SSRF allows an attacker to send crafted requests from the server side to other internal or external systems. SSRF can lead to unauthorized access to sensitive data and, in some cases, allow the attacker to control applications or systems that trust the vulnerable service. To prevent this vulnerability, avoid allowing user input to craft the base request. Instead, treat it as part of the path or query parameter and encode it appropriately. When user input is necessary to prepare the HTTP request, perform strict input validation. Additionally, whenever possible, use allowlists to only interact with expected, trusted domains.				
express-open-redirect	The application redirects to a URL specified by user-supplied input `query` that is not validated. This could redirect users to malicious locations. Consider using an allow-list approach to validate URLs, or warn users they are being redirected to a third-party website.	medium	open	main	routes/redirect.ts#L19
express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target path is nested in the intended destination	medium	open	main	routes/dataErasure.ts#L69
		medium	open	main	routes/keyServer.ts#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target path is nested in the intended destination				
express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target path is nested in the intended destination	medium	open	main	routes/logfileServer.ts#L14
express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target path is nested in the intended destination	medium	open	main	routes/quarantineServer.ts#L14
express-res-sendfile	The application processes user-input, this is passed to res.sendFile which can allow an attacker to arbitrarily read files on the system through path traversal. It is recommended to perform input validation in addition to canonicalizing the path. This allows you to validate the path against the intended directory it should be accessing.	medium	open	main	routes/fileServer.ts#L33
express-res-sendfile	The application processes user-input, this is passed to res.sendFile which can allow an attacker to arbitrarily read files on the system through path traversal. It is recommended to perform input validation in addition to canonicalizing the path. This allows you to validate the path against the intended directory it should be accessing.	medium	open	main	routes/keyServer.ts#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-res-sendfile	The application processes user-input, this is passed to res.sendFile which can allow an attacker to arbitrarily read files on the system through path traversal. It is recommended to perform input validation in addition to canonicalizing the path. This allows you to validate the path against the intended directory it should be accessing.	medium	open	main	routes/logfileServer.ts#L14
express-res-sendfile	The application processes user-input, this is passed to res.sendFile which can allow an attacker to arbitrarily read files on the system through path traversal. It is recommended to perform input validation in addition to canonicalizing the path. This allows you to validate the path against the intended directory it should be accessing.	medium	open	main	routes/quarantineServer.ts#L14
express-ssrf	The following request request.get() was found to be crafted from user-input `req` which can lead to Server-Side Request Forgery (SSRF) vulnerabilities. It is recommended where possible to not allow user-input to craft the base request, but to be treated as part of the path or query parameter. When user-input is necessary to craft the request, it is recommeneded to follow OWASP best practices to prevent abuse.	medium	open	main	routes/profileImageUrlUpload.ts#L23
express-insecure-	User data from `req` is being compiled into the template, which can lead to a	medium	open	main	routes/userProfile.ts#L56

Finding Title	Finding Description & Remediation	severity	status	ref	location
template-usage	Server Side Template Injection (SSTI) vulnerability.				
session-fixation	Detected `req` argument which enters `res.cookie`, this can lead to session fixation vulnerabilities if an attacker can control the cookie value. This vulnerability can lead to unauthorized access to accounts, and in some esoteric cases, Cross-Site-Scripting (XSS). Users should not be able to influence cookies directly, for session cookies, they should be generated securely using an approved session management library. If the cookie does need to be set by a user, consider using an allow-list based approach to restrict the cookies which can be set.	medium	open	main	lib/insecurity.ts#L195
hardcoded-jwt-secret	A hard-coded credential was detected. It is not recommended to store credentials in source-code, as this risks secrets being leaked and used by either an internal or external malicious adversary. It is recommended to use environment variables to securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).	medium	open	main	lib/insecurity.ts#L56
detect-non-literal-regexp	RegExp() called with a `challengeKey` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main	medium	open	main	lib/codingChallenges.ts#L76

Finding Title	Finding Description & Remediation	severity	status	ref	location
	thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex checking/sanitization library such as https://www.npmjs.com/package/recheck to verify that the regex does not appear vulnerable to ReDoS.				
detect-non-literal-regexp	RegExp() called with a \`file\` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex checking/sanitization library such as https://www.npmjs.com/package/recheck to verify that the regex does not appear vulnerable to ReDoS.	medium	open	main	lib/codingChallenges.ts#L76
detect-non-literal-regexp	RegExp() called with a \`paths\` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex	medium	open	main	lib/codingChallenges.ts#L76

Finding Title	Finding Description & Remediation	severity	status	ref	location
	checking/sanitization library such as https://www.npmjs.com/package/ recheck to verify that the regex does not appear vulnerable to ReDoS.				
detect-non-literal-regexp	RegExp() called with a `challengeKey` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex checking/sanitization library such as https://www.npmjs.com/package/ recheck to verify that the regex does not appear vulnerable to ReDoS.	medium	open	main	lib/codingChallenges.ts#L78
detect-non-literal-regexp	RegExp() called with a `file` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex checking/sanitization library such as https://www.npmjs.com/package/ recheck to verify that the regex does not appear vulnerable to ReDoS.	medium	open	main	lib/codingChallenges.ts#L78

Finding Title	Finding Description & Remediation	severity	status	ref	location
detect-non-literal-regexp	RegExp() called with a \`paths\` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex checking/sanitization library such as https://www.npmjs.com/package/recheck to verify that the regex does not appear vulnerable to ReDoS.	medium	open	main	lib/codingChallenges.ts#L78
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	data/staticData.ts#L7
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	lib/codingChallenges.ts#L24
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the	medium	open	main	lib/codingChallenges.ts#L24

Finding Title	Finding Description & Remediation	severity	status	ref	location
	attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.				
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	lib/startup/restoreOverwrittenFilesWithOriginals.ts#L28
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	lib/startup/validatePreconditions.ts#L120
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/dataErasure.ts#L69
path-join-resolve-traversal	Detected possible user input going into a \`path.join\` or \`path.resolve\` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/fileUpload.ts#L29

Finding Title	Finding Description & Remediation	severity	status	ref	location
path-join-resolve-traversal	Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/fileUpload.ts#L39
path-join-resolve-traversal	Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/keyServer.ts#L14
path-join-resolve-traversal	Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/logfileServer.ts#L14
path-join-resolve-traversal	Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	open	main	routes/order.ts#L45
	Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path	medium	open	main	routes/quarantineServer.ts#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
path-join-resolve-traversal	traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.				
angular-route-bypass-security-trust	Untrusted input could be used to tamper with a web page rendering, which can lead to a Cross-site scripting (XSS) vulnerability. XSS vulnerabilities occur when untrusted input executes malicious JavaScript code, leading to issues such as account compromise and sensitive information leakage. Validate the user input, perform contextual output encoding, or sanitize the input. A popular library used to prevent XSS is DOMPurify. You can also use libraries and frameworks such as Angular, Vue, and React, which offer secure defaults when rendering input.	medium	open	main	frontend/src/app/search-result/search-result.component.ts#L151
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.	medium	open	main	server.ts#L260
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If	medium	open	main	server.ts#L264

Finding Title	Finding Description & Remediation	severity	status	ref	location
	you need directory listing, ensure that sensitive files are inaccessible when querying the resource.				
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.	medium	open	main	server.ts#L268
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.	medium	open	main	server.ts#L272
hardcoded-hmac-key	Detected a hardcoded hmac key. Avoid hardcoding secrets and consider using an alternate option such as reading the secret from a config file or using an environment variable.	medium	open	main	lib/insecurity.ts#L44
hardcoded-hmac-key	Detected a hardcoded hmac key. Avoid hardcoding secrets and consider using an alternate option such as reading the secret from a config file or using an environment variable.	medium	open	main	lib/insecurity.ts#L152
		medium	open	main	frontend/src/index.html#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
missing-integrity	This tag is missing an 'integrity' subresource integrity attribute. The 'integrity' attribute allows for the browser to verify that externally hosted files (for example from a CDN) are delivered without unexpected manipulation. Without this attribute, if an attacker can modify the externally hosted resource, this could lead to XSS and other types of attacks. To prevent this, include the base64-encoded cryptographic hash of the resource (file) youâ€™re telling the browser to fetch in the 'integrity' attribute for all externally hosted files.				
missing-integrity	This tag is missing an 'integrity' subresource integrity attribute. The 'integrity' attribute allows for the browser to verify that externally hosted files (for example from a CDN) are delivered without unexpected manipulation. Without this attribute, if an attacker can modify the externally hosted resource, this could lead to XSS and other types of attacks. To prevent this, include the base64-encoded cryptographic hash of the resource (file) youâ€™re telling the browser to fetch in the 'integrity' attribute for all externally hosted files.	medium	open	main	frontend/src/index.html#L15
missing-integrity	This tag is missing an 'integrity' subresource integrity attribute. The 'integrity' attribute allows for the	medium	open	main	frontend/src/index.html#L16

Finding Title	Finding Description & Remediation	severity	status	ref	location
	browser to verify that externally hosted files (for example from a CDN) are delivered without unexpected manipulation. Without this attribute, if an attacker can modify the externally hosted resource, this could lead to XSS and other types of attacks. To prevent this, include the base64-encoded cryptographic hash of the resource (file) youâ€™re telling the browser to fetch in the 'integrity' attribute for all externally hosted files.				
eval-detected	Detected the use of eval(). eval() can be dangerous if used to evaluate dynamic content. If this content can be input from outside the program, this may be a code injection vulnerability. Ensure evaluated content is not definable by external sources.	medium	open	main	routes/captcha.ts#L23
eval-detected	Detected the use of eval(). eval() can be dangerous if used to evaluate dynamic content. If this content can be input from outside the program, this may be a code injection vulnerability. Ensure evaluated content is not definable by external sources.	medium	open	main	routes/userProfile.ts#L36
express-detect-notevil-usage	Detected usage of the \`notevil\` package, which is unmaintained and has vulnerabilities. Using any sort of \`eval()\` functionality can be very dangerous, but if you must, the \`eval\` package is an up to date alternative. Be	medium	open	main	routes/b2bOrder.ts#L22

Finding Title	Finding Description & Remediation	severity	status	ref	location
	sure that only trusted input reaches an `eval()` function.				
express-libxml-vm-noent	Detected use of parseXml() function with the `noent` field set to `true`. This can lead to an XML External Entities (XXE) attack if untrusted data is passed into it.	medium	open	main	routes/fileUpload.ts#L80
template-explicit-unescape	Detected an explicit unescape in a Pug template, using either '!=' or '!{...}'. If external data can reach these locations, your application is exposed to a cross-site scripting (XSS) vulnerability. If you must do this, ensure no external data can reach this location.	medium	open	main	views/promotionVideo.pug#L79
jssha-sha1	The SHA1 hashing algorithm is considered to be weak. If this is used in any sensitive operation such as password hashing, or is used to ensure data integrity (collision sensitive) then you should use a stronger hashing algorithm. For passwords, consider using `Argon2id`, `scrypt`, or `bcrypt`. For data integrity, consider using `SHA-256`.	medium	open	main	lib/utils.ts#L90
detected-private-key	A secret is hard-coded in the application. Secrets stored in source code, such as credentials, identifiers, and other types of sensitive data, can be leaked and used by internal or external malicious actors. Use environment variables to securely provide credentials	medium	open	main	lib/insecurity.ts#L23

Finding Title	Finding Description & Remediation	severity	status	ref	location
	and other secrets or retrieve them from a secure vault or Hardware Security Module (HSM).				
detected-private-key	A secret is hard-coded in the application. Secrets stored in source code, such as credentials, identifiers, and other types of sensitive data, can be leaked and used by internal or external malicious actors. Use environment variables to securely provide credentials and other secrets or retrieve them from a secure vault or Hardware Security Module (HSM).	medium	open	main	lib/insecurity.ts#L23
detected-private-key	A secret is hard-coded in the application. Secrets stored in source code, such as credentials, identifiers, and other types of sensitive data, can be leaked and used by internal or external malicious actors. Use environment variables to securely provide credentials and other secrets or retrieve them from a secure vault or Hardware Security Module (HSM).	medium	open	main	lib/insecurity.ts#L56
detected-private-key	A secret is hard-coded in the application. Secrets stored in source code, such as credentials, identifiers, and other types of sensitive data, can be leaked and used by internal or external malicious actors. Use environment variables to securely provide credentials and other secrets or retrieve them from	medium	open	main	lib/insecurity.ts#L152

Finding Title	Finding Description & Remediation	severity	status	ref	location
	a secure vault or Hardware Security Module (HSM).				
prototype-pollution-loop	Possibility of prototype polluting function detected. By adding or modifying attributes of an object prototype, it is possible to create attributes that exist on every object, or replace critical attributes with malicious ones. This can be problematic if the software depends on existence or non-existence of certain attributes, or uses pre-defined attributes of object prototype (such as hasOwnProperty, toString or valueOf). Possible mitigations might be: freezing the object prototype, using an object without prototypes (via Object.create(null)), blocking modifications of attributes that resolve to object prototype, using Map instead of object.	medium	open	main	frontend/src/hacking-instructor/helpers/helpers.ts#L36
unknown-value-with-script-tag	Cannot determine what 'subs' is and it is used with a '<script>' tag. This could be susceptible to cross-site scripting (XSS). Ensure 'subs' is not externally controlled, or sanitize this data.	medium	open	main	routes/videoHandler.ts#L57
unknown-value-with-script-tag	Cannot determine what 'subs' is and it is used with a '<script>' tag. This could be susceptible to cross-site scripting (XSS). Ensure 'subs' is not externally controlled, or sanitize this data.	medium	open	main	routes/videoHandler.ts#L69

Finding Title	Finding Description & Remediation	severity	status	ref	location
node-sequelize-hardcoded-secret-argument	A secret is hard-coded in the application. Secrets stored in source code, such as credentials, identifiers, and other types of sensitive data, can be leaked and used by internal or external malicious actors. Use environment variables to securely provide credentials and other secrets or retrieve them from a secure vault or Hardware Security Module (HSM).	medium	open	main	models/index.ts#L29
no-new-privileges	Service 'app' allows for privilege escalation via setuid or setgid binaries. Add 'no-new-privileges:true' in 'security_opt' to prevent this.	medium	open	main	docker-compose.test.yml#L7
writable-filesystem-service	Service 'app' is running with a writable root filesystem. This may allow malicious applications to download and run additional payloads, or modify container files. If an application inside a container has to save something temporarily consider using a tmpfs. Add 'read_only: true' to this service to prevent this.	medium	open	main	docker-compose.test.yml#L7
crlf-injection-logs	When data from an untrusted source is put into a logger and not neutralized correctly, an attacker could forge log entries or include malicious content.	medium	open	refs/pull/2/merge	src/assistant-fix-custom-message.java#L13
express-check-	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory	medium	fixed	main	server.ts#L260

Finding Title	Finding Description & Remediation	severity	status	ref	location
directory-listing	listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.				
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.	medium	fixed	main	server.ts#L264
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.	medium	fixed	main	server.ts#L268
express-check-directory-listing	Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories and files. It is recommended to disable directory listing unless it is a public resource. If you need directory listing, ensure that sensitive files are inaccessible when querying the resource.	medium	fixed	main	server.ts#L272
express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target	medium	fixed	main	routes/fileServer.ts#L33

Finding Title	Finding Description & Remediation	severity	status	ref	location
	path is nested in the intended destination				
hardcoded-hmac-key	Detected a hardcoded hmac key. Avoid hardcoding secrets and consider using an alternate option such as reading the secret from a config file or using an environment variable.	medium	fixed	main	lib/insecurity.ts#L44
hardcoded-hmac-key	Detected a hardcoded hmac key. Avoid hardcoding secrets and consider using an alternate option such as reading the secret from a config file or using an environment variable.	medium	fixed	main	lib/insecurity.ts#L152
path-join-resolve-traversal	Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.	medium	fixed	main	routes/fileServer.ts#L33

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
unsafe-formatstring	Detected string concatenation with a non-literal variable in a util.format / console.log function. If an attacker injects a format specifier in the string, it will forge the log message. Try to use constant values for the format string.	low	open	main	server.ts#L148
detect-replaceall-sanitization	Detected a call to `replaceAll()` in an attempt to HTML escape the string `tableData[i].description`. Manually sanitizing input through a manually built list can be circumvented in many situations, and it's better to use a well known sanitization library such as `sanitize-html` or `DOMPurify`.	low	open	main	data/static/codefixes/restfulXssChallenge_2.ts#L59
detect-replaceall-sanitization	Detected a call to `replaceAll()` in an attempt to HTML escape the string `tableData[i].description.replaceAll('<', '&lt;')`. Manually sanitizing input through a manually built list can be circumvented in many situations, and it's better to use a well known sanitization library such as `sanitize-html` or `DOMPurify`.	low	open	main	data/static/codefixes/restfulXssChallenge_2.ts#L59
express-check-csurf-middleware-usage	A CSRF middleware was not detected in your express application. Ensure you are either using one such as `csurf` or `csrf` (see rule references) and/or you are properly doing CSRF validation in your routes with a token or cookies.	low	open	main	server.ts#L105