Semgrep SAST Scan Report for Repository: Semgrep-Demo/secrets-demo

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	6
Findings- SAST Medium Severity	1
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
detected-aws-access-key-id-value	AWS Access Key ID Value detected. This is a sensitive credential and should not be hardcoded here. Instead, read this value from an environment variable or keep it in a separate, private file.	high	open	main	test.txt#L8
detected-aws-secret-access-key	AWS Secret Access Key detected	high	open	main	test.txt#L9
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/13/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	open	refs/pull/13/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 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	refs/pull/12/merge	src/assistant-fix-sqli-sequelize.ts#L5
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	high	open	refs/pull/12/merge	src/assistant-fix-sqli-sequelize.ts#L5

Finding Title	Finding Description & Remediation	severity	status	ref	location
	SQL injection, it is recommended to use parameterized queries or prepared statements.				

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/13/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/ruby-app

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	4
Findings- SAST Medium Severity	1
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/3/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	open	refs/pull/3/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 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	refs/pull/2/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/2/merge	src/assistant-fix-sqli-sequelize.ts#L5

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/3/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/yet-another-service

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	0
Findings- SAST Medium Severity	0
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/supply-chain-demo

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	4
Findings- SAST Medium Severity	1
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/10/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	open	refs/pull/10/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 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	refs/pull/9/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/9/merge	src/assistant-fix-sqli-sequelize.ts#L5

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/10/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/python-app

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	13
Findings- SAST Medium Severity	27
Findings- SAST Low Severity	3

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
generic-sql-flask	Untrusted input might be used to build a database query, which can lead to a SQL injection vulnerability. An attacker can execute malicious SQL statements and gain unauthorized access to sensitive data, modify, delete data, or execute arbitrary system commands. The driver API has the ability to bind parameters to the query in a safe way. Make sure not to dynamically create SQL queries from user-influenced inputs. If you cannot avoid this, either escape the data properly or create an allowlist to check the value.	high	open	main	app/app.py#L265
tainted-pyyaml-flask	The application may convert user-controlled data into an object, which can lead to an insecure deserialization vulnerability. An attacker can create a malicious serialized object, pass it to the application, and take advantage of the deserialization process to perform Denial-of-service (DoS), Remote code execution (RCE), or bypass access control measures. PyYAML's `yaml` module is as powerful as `pickle` and so may call auny Python function. It is recommended to secure your application by using `yaml.SafeLoader` or `yaml.CSafeLoader`.	high	open	main	app/app.py#L329
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 the Django object-relational mappers (ORM) instead of raw SQL queries.	high	open	main	app/app.py#L261

Finding Title	Finding Description & Remediation	severity	status	ref	location
dangerous-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	high	open	main	app/app.py#L103
dangerous-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	high	open	main	app/app.py#L271
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 SQLAlchemy which will protect your queries.	high	open	main	app/app.py#L261
insecure-deserialization	Detected the use of an insecure deserialization library in a Flask route. These libraries are prone to code execution vulnerabilities. Ensure user data does not enter this function. To fix this, try to avoid serializing whole objects. Consider instead using a serializer such as JSON.	high	open	main	app/app.py#L329
jwt-python-hardcoded-secret	Hardcoded JWT secret or private key is used. This is a Insufficiently Protected Credentials weakness: https://cwe.mitre.org/data/definitions/522.html Consider using an appropriate security mechanism to protect the credentials (e.g. keeping secrets in environment variables)	high	open	main	app/app.py#L184
sqlalchemy-execute-raw-query	Avoiding SQL string concatenation: untrusted input concatenated with raw SQL query can result in SQL Injection. In order to execute raw query safely, prepared statement should be used. SQLAlchemy provides TextualSQL to easily used prepared statement with named parameters. For complex SQL composition, use SQL Expression Language or Schema	high	open	main	app/app.py#L265

Finding Title	Finding Description & Remediation	severity	status	ref	location
	Definition Language. In most cases, SQLAlchemy ORM will be a better option.				
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/16/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	open	refs/pull/16/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 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	refs/pull/15/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/15/merge	src/assistant-fix-sqli-sequelize.ts#L5

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
raw-html-format	Detected user input flowing into a manually constructed HTML string. You may be accidentally bypassing secure methods of rendering HTML by manually constructing HTML and this could create a cross-site scripting vulnerability, which could let attackers steal sensitive user data. To be sure this is safe, check that the HTML is rendered safely. Otherwise, use templates (`django.shortcuts.render`) which will safely render HTML instead.	medium	open	main	app/app.py#L103
render-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	medium	open	main	app/app.py#L114
render-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	medium	open	main	app/app.py#L281
raw-html-format	Detected user input flowing into a manually constructed HTML string. You may be accidentally bypassing secure methods of rendering HTML by manually constructing HTML and this could create a cross-site scripting vulnerability, which could let attackers steal sensitive user data. To be sure this is safe, check that the HTML is rendered safely. Otherwise, use templates (`flask.render_template`) which will safely render HTML instead.	medium	open	main	app/app.py#L103
formatted-sql-query	Detected possible formatted SQL query. Use parameterized queries instead.	medium	open	main	app/app.py#L265
md5-used-as-password	It looks like MD5 is used as a password hash. MD5 is not considered a secure password hash because it can be cracked by	medium	open	main	app/app.py#L141

Finding Title	Finding Description & Remediation	severity	status	ref	location
	an attacker in a short amount of time. Use a suitable password hashing function such as scrypt. You can use `hashlib.scrypt`.				
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	main	app/app.py#L326
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L148
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L167
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L192
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L194
client-error-return	Error return (code 403) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L200
client-error-return	Error return (code 403) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L203
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L216

Finding Title	Finding Description & Remediation	severity	status	ref	location
client-error-return	Error return (code 400) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L218
client-error-return	Error return (code 403) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L225
client-error-return	Error return (code 403) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L228
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L239
client-error-return	Error return (code 400) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L241
client-error-return	Error return (code 403) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L250
client-error-return	Error return (code 403) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L253
client-error-return	Error return (code 404) detected. This bypasses our error-handling framework. You should instead raise the relevant error from werkzeug.exceptions().\n	medium	open	main	app/app.py#L281
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using	medium	open	main	app/templates/index.html#L12

Finding Title	Finding Description & Remediation	severity	status	ref	location
	a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	app/templates/index.html#L12
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use 'url_for()' to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	app/templates/index.html#L12
third-party-action-not-pinned-to-commit-sha	An action sourced from a third-party repository on GitHub is not pinned to a full length commit SHA. Pinning an action to a full length commit SHA is currently the only way to use an action as an immutable release. Pinning to a particular SHA helps mitigate the risk of a bad actor adding a backdoor to the action's repository, as they would need to generate a SHA-1 collision for a valid Git object payload.	medium	open	main	old-workflows/semgrep.yml#L12
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/16/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
flask-use-jsonify-secure-default	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. To prevent this vulnerability, validate the user input, perform contextual output encoding or sanitize the input. In Flask apps, it is recommended to use the `jsonify()` function instead of the `json.dumps()` functions. It is more convenient as it converts the JSON data to a Response object, using `json.dumps()` is more error prone. Additionally, `jsonify()` sets the correct security headers and the response type for JSON responses. This means the response data will never be interpreted by browsers as HTML or JavaScript and will be secure against XSS attacks.	low	open	main	app/app.py#L190
flask-use-jsonify-secure-default	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. To prevent this vulnerability, validate the user input, perform contextual output encoding or sanitize the input. In Flask apps, it is recommended to use the `jsonify()` function instead of the `json.dumps()` functions. It is more convenient as it converts the JSON data to a Response object, using `json.dumps()` is more error prone. Additionally, `jsonify()` sets the correct security headers and the response type for JSON responses. This means the response data will never be interpreted by browsers as HTML or JavaScript and will be secure against XSS attacks.	low	fixed	main	app/app.py#L185
flask-use-jsonify-	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. To prevent this	low	fixed	main	app/app.py#L331

Finding Title	Finding Description & Remediation	severity	status	ref	location
secure-default	vulnerability, validate the user input, perform contextual output encoding or sanitize the input. In Flask apps, it is recommended to use the `jsonify()` function instead of the `json.dumps()` functions. It is more convenient as it converts the JSON data to a Response object, using `json.dumps()` is more error prone. Additionally, `jsonify()` sets the correct security headers and the response type for JSON responses. This means the response data will never be interpreted by browsers as HTML or JavaScript and will be secure against XSS attacks.				

Semgrep SAST Scan Report for Repository: Semgrep-Demo/Cesar-JsGithubAPI

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	0
Findings- SAST Medium Severity	4
Findings- SAST Low Severity	1

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

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	githubWorkflow.js#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	githubWorkflow.js#L103
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	githubWorkflow.js#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	githubWorkflow.js#L162

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	githubWorkflow.js#L187

Semgrep SAST Scan Report for Repository: local_scan/secrets-testing

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	0
Findings- SAST Medium Severity	1
Findings- SAST Low Severity	2

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
symmetric-hardcoded-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	secrets	cry.js#L2

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	secrets	server.js#L2
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	secrets	server.js#L10

Semgrep SAST Scan Report for Repository: Semgrep-Demo/ JavaLog4J

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	**Vulnerability Count**
Findings- SAST High Severity	7
Findings- SAST Medium Severity	0
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	main	vulnerable-application/src/main/java/com/example/log4shell/LoginServlet.java#L34
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	main	vulnerable-application/src/main/java/com/example/log4shell/LoginServlet.java#L35
dangerous-subprocess-use-audit	Detected subprocess function 'run' without a static string. If this data can be controlled by a malicious actor, it may be an instance of command injection. Audit the use of this call to ensure it is not controllable by an external resource. You may consider using 'shlex.escape()'.	high	open	main	poc.py#L62
dangerous-subprocess-use-audit	Detected subprocess function 'call' without a static string. If this data can be controlled by a malicious actor, it may be an instance of command injection. Audit the use of this call to ensure it is not controllable by an external resource. You may consider using 'shlex.escape()'.	high	open	main	poc.py#L86
dangerous-subprocess-use-audit	Detected subprocess function 'run' without a static string. If this data can be controlled by a malicious actor, it may be an instance of command injection. Audit the use of this call to ensure it is not controllable by an external	high	open	main	poc.py#L98

Finding Title	Finding Description & Remediation	severity	status	ref	location
	resource. You may consider using 'shlex.escape()'.				
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	main	vulnerable-application/src/main/java/com/example/log4shell/LoginServlet.java#L34
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	open	main	vulnerable-application/src/main/java/com/example/log4shell/LoginServlet.java#L35

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/ supply-chain-second-demo

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	4
Findings- SAST Medium Severity	1
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	refs/pull/4/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/4/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/5/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	open	refs/pull/5/merge	src/assistant-fix-custom-message.java#L14

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/5/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/new-project

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	12
Findings- SAST Medium Severity	21
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
grpc-server-insecure-connection	Found an insecure gRPC server without 'grpc.Creds()' or options with credentials. This allows for a connection without encryption to this server. A malicious attacker could tamper with the gRPC message, which could compromise the machine. Include credentials derived from an SSL certificate in order to create a secure gRPC connection. You can create credentials using 'credentials.NewServerTLSFromFile("cert.pem", "cert.key")'.	high	reviewing	main	src/shippingservice/main.go#L85
grpc-server-insecure-connection	Found an insecure gRPC server without 'grpc.Creds()' or options with credentials. This allows for a connection without encryption to this server. A malicious attacker could tamper with the gRPC message, which could compromise the machine. Include credentials derived from an SSL certificate in order to create a secure gRPC connection. You can create credentials using 'credentials.NewServerTLSFromFile("cert.pem", "cert.key")'.	high	open	main	src/shippingservice/main.go#L88
missing-user-entrypoint	By not specifying a USER, a program in the container may run as 'root'. This is a security hazard. If an attacker can control a process running as root, they may have control over the container. Ensure that the last USER in a Dockerfile is a USER other than 'root'.	high	open	main	src/loadgenerator/Dockerfile#L35
	Found an insecure gRPC connection. This creates a connection without encryption to a gRPC client/	high	open	main	src/currencyservice/client.js#L21

Finding Title	Finding Description & Remediation	severity	status	ref	location
grpc-nodejs-insecure-connection	server. A malicious attacker could tamper with the gRPC message, which could compromise the machine.				
arbitrary-sleep	time.sleep() call; did you mean to leave this in?	high	open	main	src/emailservice/email_server.py#L134
arbitrary-sleep	time.sleep() call; did you mean to leave this in?	high	open	main	src/emailservice/email_server.py#L158
arbitrary-sleep	time.sleep() call; did you mean to leave this in?	high	open	main	src/recommendationservice/recommendation_server.py#L61
arbitrary-sleep	time.sleep() call; did you mean to leave this in?	high	open	main	src/recommendationservice/recommendation_server.py#L151
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	open	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 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-	high	open	refs/pull/1/merge	src/assistant-fix-sqli-sequelize.ts#L5

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	src/paymentservice/server.js#L37
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	src/paymentservice/server.js#L38
math-random-used	Do not use `math/rand`. Use `crypto/rand` instead.	medium	open	main	src/frontend/handlers.go#L21
math-random-used	Do not use `math/rand`. Use `crypto/rand` instead.	medium	open	main	src/shippingservice/tracker.go#L19
cookie-missing-httponly	A session cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie which mitigates XSS attacks. Set the 'HttpOnly' flag by setting 'HttpOnly' to 'true' in the Cookie.	medium	open	main	src/frontend/handlers.go#L418
cookie-missing-httponly	A session cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie which mitigates XSS attacks. Set the 'HttpOnly' flag by setting 'HttpOnly' to 'true' in the Cookie.	medium	open	main	src/frontend/middleware.go#L97

Finding Title	Finding Description & Remediation	severity	status	ref	location
cookie-missing-secure	A session cookie was detected without setting the 'Secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'Secure' flag by setting 'Secure' to 'true' in the Options struct.	medium	open	main	src/frontend/handlers.go#L418
cookie-missing-secure	A session cookie was detected without setting the 'Secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'Secure' flag by setting 'Secure' to 'true' in the Options struct.	medium	open	main	src/frontend/middleware.go#L97
use-tls	Found an HTTP server without TLS. Use 'http.ListenAndServeTLS' instead. See https://golang.org/pkg/net/http/#ListenAndServeTLS for more information.	medium	open	main	src/frontend/main.go#L159
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	src/frontend/templates/ad.html#L21
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	src/frontend/templates/ad.html#L21
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/frontend/templates/cart.html#L45

Finding Title	Finding Description & Remediation	severity	status	ref	location
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/frontend/templates/header.html#L71
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/frontend/templates/product.html#L38
direct-use-of-jinja2	Detected direct use of jinja2. If not done properly, this may bypass HTML escaping which opens up the application to cross-site scripting (XSS) vulnerabilities. Prefer using the Flask method 'render_template()' and templates with a '.html' extension in order to prevent XSS.	medium	open	main	src/emailservice/email_server.py#L45
direct-use-of-jinja2	Detected direct use of jinja2. If not done properly, this may bypass HTML escaping which opens up the application to cross-site scripting (XSS) vulnerabilities. Prefer using the Flask method 'render_template()' and templates with a '.html' extension in order to prevent XSS.	medium	open	main	src/emailservice/email_server.py#L90
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use 'url_for()' to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	src/frontend/templates/ad.html#L21
string-to-int-signedness-cast	Downcasting or changing sign of an integer with \`$CAST_METHOD\` method	medium	open	main	src/frontend/handlers.go#L214
context-todo	Consider to use well-defined context	medium	open	main	src/checkoutservice/main.go#L357
	These functions do not allow to set a a timeout value for reading requests. As a result, the app server may be vulnerable to a Slowloris Denial-of-Service (DoS) attack. Slowloris	medium	open	main	src/frontend/main.go#L159

Finding Title	Finding Description & Remediation	severity	status	ref	location
slowloris-dos-functions	attacks exploit the fact that HTTP servers keep the connection active if the request received is incomplete. To mitigate this, implement a `Server` and set the timeout with `ReadHeaderTimeout`.				
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

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/js-app

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	27
Findings- SAST Medium Severity	71
Findings- SAST Low Severity	3

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 sanitize the data if you	high	open	main	routes/likeProductReviews.ts#L25

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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/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#L18

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/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 mapper (ORM) such as Sequelize which will protect your queries.	high	open	main	data/static/codefixes/dbSchemaChallenge_1.ts#L5
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	high	open	main	data/static/codefixes/unionSqlInjectionChallenge_1.ts#L6

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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.				
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_3.ts#L10
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	high	open	main	routes/search.ts#L23

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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.				
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_1.ts#L5
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
		high	open	main	

Finding Title	Finding Description & Remediation	severity	status	ref	location
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.				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 recommended to use parameterized queries or prepared statements.	high	open	main	routes/login.ts#L36
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	high	open	refs/pull/	src/assistant-fix-custom-message.java#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
	attacker could forge log entries or include malicious content.			34/ merge	
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	open	refs/ pull/ 34/ 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 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	refs/ pull/ 33/ merge	src/assistant-fix-sqli-sequelize.ts#L5
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/ 33/ 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 files. If the application does not validate user	medium	open	main	routes/profileImageUrlUpload.ts#L31

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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#L81
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	medium	open	main	routes/vulnCodeFixes.ts#L82

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	open	main	routes/redirect.ts#L19
regexp-redos	Detected \`req\` argument enters calls to \`RegExp\`. This could lead to a Regular Expression Denial of Service (ReDoS) through catastrophic backtracking. If the input is attacker controllable, this vulnerability can lead to systems being	medium	open	main	routes/vulnCodeSnippet.ts#L104

Finding Title	Finding Description & Remediation	severity	status	ref	location
	non-responsive or may crash due to ReDoS. Where possible avoid calls to `RegExp` with user input, if required ensure user input is escaped or validated.				
regexp-redos	Detected `req` argument enters calls to `RegExp`. This could lead to a Regular Expression Denial of Service (ReDoS) through catastrophic backtracking. If the input is attacker controllable, this vulnerability can lead to systems being non-responsive or may crash due to ReDoS. Where possible avoid calls to `RegExp` with user input, if required ensure user input is escaped or validated.	medium	open	main	routes/vulnCodeSnippet.ts#L108
regexp-redos	Detected `req` argument enters calls to `RegExp`. This could lead to a Regular Expression Denial of Service (ReDoS) through catastrophic backtracking. If the input is attacker controllable, this vulnerability can lead to systems being non-responsive or may crash due to ReDoS. Where possible avoid calls to `RegExp` with user input, if required ensure user input is escaped or validated.	medium	open	main	routes/vulnCodeSnippet.ts#L123
regexp-redos	Detected `req` argument enters calls to `RegExp`. This could lead to a Regular Expression Denial of Service (ReDoS) through catastrophic backtracking. If the input is attacker controllable, this vulnerability can lead to systems being non-responsive or may crash due to ReDoS. Where possible avoid calls to	medium	open	main	routes/vulnCodeSnippet.ts#L125

Finding Title	Finding Description & Remediation	severity	status	ref	location
	`RegExp` with user input, if required ensure user input is escaped or validated.				
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.	medium	open	main	routes/profileImageUrlUpload.ts#L23
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

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	medium	open	main	routes/keyServer.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/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
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	medium	open	main	routes/logfileServer.ts#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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.				
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-template-usage	User data from `req` is being compiled into the template, which can lead to a Server Side Template Injection (SSTI) vulnerability.	medium	open	main	routes/userProfile.ts#L56
		medium	open	main	lib/insecurity.ts#L211

Finding Title	Finding Description & Remediation	severity	status	ref	location
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.				
detect-non-literal-regexp	RegExp() called with a `key` 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	routes/vulnCodeSnippet.ts#L104
detect-non-literal-regexp	RegExp() called with a `req` 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	medium	open	main	routes/vulnCodeSnippet.ts#L104

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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 `key` 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	routes/vulnCodeSnippet.ts#L123
detect-non-literal-regexp	RegExp() called with a `req` 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	routes/vulnCodeSnippet.ts#L123

Finding Title	Finding Description & Remediation	severity	status	ref	location
detect-non-literal-regexp	RegExp() called with a `key` 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	routes/vulnCodeSnippet.ts#L125
detect-non-literal-regexp	RegExp() called with a `req` 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	routes/vulnCodeSnippet.ts#L125
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	medium	open	main	data/datacreator.ts#L41

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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#L30
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#L95
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#L46
	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.				
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/vulnCodeSnippet.ts#L33
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/vulnCodeSnippet.ts#L33
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/vulnCodeSnippet.ts#L107
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	medium	open	main	frontend/src/app/search-result/search-result.component.ts#L152

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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.				
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#L241
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#L246
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#L250
		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 browser to verify that externally hosted files (for example from a CDN) are delivered	medium	open	main	frontend/src/index.html#L16

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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 sure that only trusted input reaches an `eval()` function.	medium	open	main	routes/b2bOrder.ts#L22
		medium	open	main	lib/insecurity.ts#L53

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-jwt-not-revoked	No token revoking configured for `express-jwt`. A leaked token could still be used and unable to be revoked. Consider using function as the `isRevoked` option.				
express-jwt-not-revoked	No token revoking configured for `express-jwt`. A leaked token could still be used and unable to be revoked. Consider using function as the `isRevoked` option.	medium	open	main	lib/insecurity.ts#L54
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
jwt-exposed-data	The object is passed strictly to jsonwebtoken.sign(...) Make sure that sensitive information is not exposed through JWT token payload.	medium	open	main	lib/insecurity.ts#L55
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	medium	open	main	lib/utils.ts#L97

Finding Title	Finding Description & Remediation	severity	status	ref	location
	should use a stronger hashing algorithm. For passwords, consider using `Argon2id`, `scrypt`, or `bcrypt`. For data integrity, consider using `SHA-256`.				
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#L43
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

Finding Title	Finding Description & Remediation	severity	status	ref	location
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
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#L31
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	medium	open	refs/ pull/	src/assistant-fix-custom-message.java#L13

Finding Title	Finding Description & Remediation	severity	status	ref	location
	correctly, an attacker could forge log entries or include malicious content.			34/ merge	
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#L241
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#L246
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#L250
express-path-join-resolve-traversal	Possible writing outside of the destination, make sure that the target path is nested in the intended destination	medium	fixed	main	routes/fileServer.ts#L33
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	medium	fixed	main	routes/fileServer.ts#L33

Finding Title	Finding Description & Remediation	severity	status	ref	location
	can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.				

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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#L91

Semgrep SAST Scan Report for Repository: Semgrep-Demo/go-app

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	10
Findings- SAST Medium Severity	60
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
remote-property-injection	Bracket object notation with user input is present, this might allow an attacker to access all properties of the object and even it's prototype. Use literal values for object properties.	high	open	main	public/js/jquery-3.2.1-min.js#L4
gosql-sqli	Detected string concatenation with a non-literal variable in a "database/sql" Go SQL statement. This could lead to SQL injection if the variable is user-controlled and not properly sanitized. In order to prevent SQL injection, use parameterized queries or prepared statements instead. You can use prepared statements with the 'Prepare' and 'PrepareContext' calls.	high	open	main	util/database/database.go#L24
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	high	open	main	templates/template.header.html#L13
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template	high	open	main	templates/template.header.html#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
	tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).				
err-nil-check	superfluous nil err check before return	high	open	main	setup/function.go#L58
err-nil-check	superfluous nil err check before return	high	open	main	setup/function.go#L74
err-nil-check	superfluous nil err check before return	high	open	main	vulnerability/idor/function.go#L50
err-nil-check	superfluous nil err check before return	high	open	main	vulnerability/idor/function.go#L76
err-nil-check	superfluous nil err check before return	high	open	main	vulnerability/sqli/function.go#L76
deprecated-ioutil-readfile	ioutil.ReadFile is deprecated	high	open	main	util/config/config.go#L24

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
session-cookie-missing-httponly	A session cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie which mitigates XSS attacks. Set the 'HttpOnly' flag by setting 'HttpOnly' to 'true' in the Options struct.	medium	open	main	user/session/session.go#L28
session-cookie-missing-httponly	A session cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie which mitigates XSS attacks. Set the 'HttpOnly' flag by setting 'HttpOnly' to 'true' in the Options struct.	medium	open	main	user/session/session.go#L67
session-cookie-missing-secure	A session cookie was detected without setting the 'Secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'Secure' flag by setting 'Secure' to 'true' in the Options struct.	medium	open	main	user/session/session.go#L28
session-cookie-missing-secure	A session cookie was detected without setting the 'Secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'Secure' flag by setting 'Secure' to 'true' in the Options struct.	medium	open	main	user/session/session.go#L67
use-of-md5	Detected MD5 hash algorithm which is considered insecure. MD5 is not collision resistant and is therefore not suitable as a cryptographic signature. Use SHA256 or SHA3 instead.	medium	open	main	user/user.go#L160
use-of-md5	Detected MD5 hash algorithm which is considered insecure. MD5 is not collision resistant and is therefore not	medium	open	main	vulnerability/csa/csa.go#L62

Finding Title	Finding Description & Remediation	severity	status	ref	location
	suitable as a cryptographic signature. Use SHA256 or SHA3 instead.				
use-of-md5	Detected MD5 hash algorithm which is considered insecure. MD5 is not collision resistant and is therefore not suitable as a cryptographic signature. Use SHA256 or SHA3 instead.	medium	open	main	vulnerability/idor/idor.go#L164
string-formatted-query	String-formatted SQL query detected. This could lead to SQL injection if the string is not sanitized properly. Audit this call to ensure the SQL is not manipulable by external data.	medium	open	main	util/database/database.go#L24
string-formatted-query	String-formatted SQL query detected. This could lead to SQL injection if the string is not sanitized properly. Audit this call to ensure the SQL is not manipulable by external data.	medium	open	main	vulnerability/sqli/function.go#L37
cookie-missing-httponly	A session cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie which mitigates XSS attacks. Set the 'HttpOnly' flag by setting 'HttpOnly' to 'true' in the Cookie.	medium	open	main	util/cookie.go#L32
cookie-missing-httponly	A session cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie which mitigates XSS attacks. Set the 'HttpOnly' flag by setting 'HttpOnly' to 'true' in the Cookie.	medium	open	main	util/cookie.go#L48
cookie-missing-secure	A session cookie was detected without setting the 'Secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'Secure' flag by setting 'Secure' to 'true' in the Options struct.	medium	open	main	util/cookie.go#L32

Finding Title	Finding Description & Remediation	severity	status	ref	location
cookie-missing-secure	A session cookie was detected without setting the 'Secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'Secure' flag by setting 'Secure' to 'true' in the Options struct.	medium	open	main	util/cookie.go#L48
formatted-template-string	Found a formatted template string passed to 'template.HTML()'. 'template.HTML()' does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability.	medium	open	main	vulnerability/xss/xss.go#L52
formatted-template-string	Found a formatted template string passed to 'template.HTML()'. 'template.HTML()' does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability.	medium	open	main	vulnerability/xss/xss.go#L53
formatted-template-string	Found a formatted template string passed to 'template.HTML()'. 'template.HTML()' does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability.	medium	open	main	vulnerability/xss/xss.go#L61
formatted-template-string	Found a formatted template string passed to 'template.HTML()'. 'template.HTML()' does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability.	medium	open	main	vulnerability/xss/xss.go#L96
no-direct-write-to-responsewriter	Detected directly writing or similar in 'http.ResponseWriter.write()'. This bypasses HTML escaping that prevents cross-site scripting vulnerabilities.	medium	open	main	util/template.go#L35

Finding Title	Finding Description & Remediation	severity	status	ref	location
	Instead, use the 'html/template' package and render data using 'template.Execute()'.				
unsafe-template-type	Semgrep could not determine that the argument to 'template.HTML()' is a constant. 'template.HTML()' and similar does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability. Instead, do not use this function and use 'template.Execute()'.	medium	open	main	util/template.go#L45
unsafe-template-type	Semgrep could not determine that the argument to 'template.HTML()' is a constant. 'template.HTML()' and similar does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability. Instead, do not use this function and use 'template.Execute()'.	medium	open	main	vulnerability/xss/xss.go#L58
unsafe-template-type	Semgrep could not determine that the argument to 'template.HTML()' is a constant. 'template.HTML()' and similar does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability. Instead, do not use this function and use 'template.Execute()'.	medium	open	main	vulnerability/xss/xss.go#L59
unsafe-template-type	Semgrep could not determine that the argument to 'template.HTML()' is a constant. 'template.HTML()' and similar does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability. Instead, do not use this function and use 'template.Execute()'.	medium	open	main	vulnerability/xss/xss.go#L62

Finding Title	Finding Description & Remediation	severity	status	ref	location
unsafe-template-type	Semgrep could not determine that the argument to 'template.HTML()' is a constant. 'template.HTML()' and similar does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability. Instead, do not use this function and use 'template.Execute()'.	medium	open	main	vulnerability/xss/xss.go#L63
unsafe-template-type	Semgrep could not determine that the argument to 'template.HTML()' is a constant. 'template.HTML()' and similar does not escape contents. Be absolutely sure there is no user-controlled data in this template. If user data can reach this template, you may have a XSS vulnerability. Instead, do not use this function and use 'template.Execute()'.	medium	open	main	vulnerability/xss/xss.go#L100
raw-html-format	Detected user input flowing into a manually constructed HTML string. You may be accidentally bypassing secure methods of rendering HTML by manually constructing HTML and this could create a cross-site scripting vulnerability, which could let attackers steal sensitive user data. Use the `html/template` package which will safely render HTML instead, or inspect that the HTML is rendered safely.	medium	open	main	vulnerability/xss/xss.go#L96
formatted-template-string-taint	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. To prevent this vulnerability, validate the user input, perform contextual output encoding or sanitize the input. For more information, see: [Go XSS prevention](https://semgrep.dev/docs/cheat-sheets/go-xss/).	medium	open	main	vulnerability/xss/xss.go#L100

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.	medium	open	main	templates/cart.html#L7
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	templates/template.login.html#L7
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	templates/template.login.html#L8
		medium	open	main	

Finding Title	Finding Description & Remediation	severity	status	ref	location
plaintext-http-link	This link points to a plaintext HTTP URL. Prefer an encrypted HTTPS URL if possible.				templates/template.idor1.html#L56
plaintext-http-link	This link points to a plaintext HTTP URL. Prefer an encrypted HTTPS URL if possible.	medium	open	main	templates/template.idor2.html#L57
plaintext-http-link	This link points to a plaintext HTTP URL. Prefer an encrypted HTTPS URL if possible.	medium	open	main	templates/template.sqli1.html#L19
plaintext-http-link	This link points to a plaintext HTTP URL. Prefer an encrypted HTTPS URL if possible.	medium	open	main	templates/template.sqli2.html#L18
plaintext-http-link	This link points to a plaintext HTTP URL. Prefer an encrypted HTTPS URL if possible.	medium	open	main	templates/template.sqli2.html#L19
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/setup.sidebar.html#L12
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/setup.sidebar.html#L18
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward	medium	open	main	templates/template.sidebar.html#L12

Finding Title	Finding Description & Remediation	severity	status	ref	location
	slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L18
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L27
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L28
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L36

Finding Title	Finding Description & Remediation	severity	status	ref	location
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L37
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L44
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L45
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L56
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS)	medium	open	main	templates/ template.sidebar.html#L63

Finding Title	Finding Description & Remediation	severity	status	ref	location
	attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/template.sidebar.html#L68
detect-non-literal-regexp	RegExp() called with a \`a\` 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	public/js/jquery-3.2.1-min.js#L2
detect-non-literal-regexp	RegExp() called with a \`b\` 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	public/js/jquery-3.2.1-min.js#L3
detect-non-literal-regexp	RegExp() called with a \`b\` function argument, this might allow an attacker to cause a Regular Expression Denial-of-	medium	open	main	public/js/jquery-3.2.1-min.js#L4

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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.				
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	public/js/jquery-3.2.1-min.js#L2
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	public/js/jquery-3.2.1-min.js#L2
		medium	open	main	

Finding Title	Finding Description & Remediation	severity	status	ref	location
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.				templates/ template.sidebar.html#L28
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L37
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	templates/ template.login.html#L29
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use 'url_for()' to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L28
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use 'url_for()' to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	main	templates/ template.sidebar.html#L37
no-new-privileges	Service 'db-mysql' 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.yml#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
writable-filesystem-service	Service 'db-mysql' 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.yml#L14
formatted-template-string-taint-copy	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. To prevent this vulnerability, validate the user input, perform contextual output encoding or sanitize the input. For more information, see: [Go XSS prevention](https://semgrep.dev/docs/cheat-sheets/go-xss/).	medium	open	main	vulnerability/xss/xss.go#L100

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: local_scan/vulnado

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	37
Findings- SAST Medium Severity	59
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
command-injection-process-builder	A formatted or concatenated string was detected as input to a ProcessBuilder call. This is dangerous if a variable is controlled by user input and could result in a command injection. Ensure your variables are not controlled by users or sufficiently sanitized.	high	open	master	src/main/java/com/scalesec/vulnado/Cowsay.java#L11
formatted-sql-string-deepsemgrep	Untrusted input might be used to build a database query, which can lead to a SQL injection vulnerability. An attacker can execute malicious SQL statements and gain unauthorized access to sensitive data, modify, delete data, or execute arbitrary system commands. To prevent this vulnerability, use prepared statements that do not concatenate user-controllable strings and use parameterized queries where SQL commands and user data are strictly separated. Also, consider using an object-relational (ORM) framework to operate with safer abstractions. To build SQL queries safely in Java, it is possible to adopt prepared statements by using the `java.sql.PreparedStatement` class with bind variables.	high	open	master	src/main/java/com/scalesec/vulnado/User.java#L49
formatted-sql-string	Detected a formatted string in a SQL statement. This could lead to SQL injection if variables in the SQL statement are not properly sanitized. Use a prepared statements (java.sql.PreparedStatement) instead. You can obtain a PreparedStatement using 'connection.prepareStatement'.	high	open	master	src/main/java/com/scalesec/vulnado/User.java#L49
tainted-system-command	Untrusted input might be injected into a command executed by the application, which can lead to a command injection vulnerability. An attacker can execute arbitrary commands, potentially gaining complete control of the system. To prevent this vulnerability, avoid executing OS commands with user input. If this is unavoidable, validate and sanitize the input, and	high	open	master	src/main/java/com/scalesec/vulnado/Cowsay.java#L11

Finding Title	Finding Description & Remediation	severity	status	ref	location
	use safe methods for executing the commands. For more information, see: [Java command injection prevention](https://semgrep.dev/docs/cheat-sheets/java-command-injection/)				
tainted-sql-string	User data flows into this manually-constructed SQL string. User data can be safely inserted into SQL strings using prepared statements or an object-relational mapper (ORM). Manually-constructed SQL strings is a possible indicator of SQL injection, which could let an attacker steal or manipulate data from the database. Instead, use prepared statements (`connection.PreparedStatement`) or a safe library.	high	open	master	src/main/java/com/scalesec/vulnado/User.java#L47
tainted-url-host	User data flows into the host portion of this manually-constructed URL. This could allow an attacker to send data to their own server, potentially exposing sensitive data such as cookies or authorization information sent with this request. They could also probe internal servers or other resources that the server runnig this code can access. (This is called server-side request forgery, or SSRF.) Do not allow arbitrary hosts. Instead, create an allowlist for approved hosts hardcode the correct host, or ensure that the user data can only affect the path or parameters.	high	open	master	src/main/java/com/scalesec/vulnado/LinkLister.java#L26
tainted-ssrf-spring-add	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.	high	open	master	src/main/java/com/scalesec/vulnado/LinkLister.java#L16

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	master	sqli.js#L35
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	master	sqli.js#L38
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	master	sqli.js#L41
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	high	open	master	sqli.js#L44

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	master	sqli.js#L47
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	master	sqli.js#L50
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	master	sqli.js#L53
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	high	open	master	v-sqli.js#L34

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	master	v-sqli.js#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	master	v-sqli.js#L39
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	master	vv-sqli.js#L34
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.	high	open	master	vv-sqli.js#L36

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	master	vv-sqli.js#L39
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	master	sqli.js#L35
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	master	sqli.js#L38
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	master	sqli.js#L41
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	master	sqli.js#L44

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	master	sqli.js#L47
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	master	sqli.js#L50
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	master	sqli.js#L53
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	master	v-sqli.js#L34
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	master	v-sqli.js#L36
	Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if the variable is user-controlled and	high	open	master	v-sqli.js#L39

Finding Title	Finding Description & Remediation	severity	status	ref	location
express-sequelize-injection	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	master	vv-sqli.js#L34
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	master	vv-sqli.js#L36
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	master	vv-sqli.js#L39
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	high	open	master	client/index.html#L63
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not	high	open	master	client/index.html#L67

Finding Title	Finding Description & Remediation	severity	status	ref	location
	always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., \`document.getElementById\`).				
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., \`document.getElementById\`).	high	open	master	client/index.html#L67
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., \`document.getElementById\`).	high	open	master	client/index.html#L73

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
cookie-missing-secure-flag	A cookie was detected without setting the 'secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'secure' flag by calling 'cookie.setSecure(true);'	medium	open	master	vulns/newnewnenwbadcookie.java#L23
cookie-secure-flag-false	A cookie was detected without setting the 'secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'secure' flag by calling 'cookie.setSecure(true);'	medium	open	master	vulns/newnewnenwbadcookie.java#L23
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	medium	open	master	client/index.html#L63
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion	medium	open	master	client/index.html#L67

Finding Title	Finding Description & Remediation	severity	status	ref	location
	(outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).				
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	medium	open	master	client/index.html#L67
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	medium	open	master	client/index.html#L73
missing-integrity	This tag is missing an 'integrity' subresource integrity attribute. The 'integrity' attribute allows	medium	open	master	client/index.html#L57

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	master	client/index.html#L60
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	medium	open	master	client/login.html#L40

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	master	client/login.html#L43
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L10
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L19
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L28
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L33
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L35

Finding Title	Finding Description & Remediation	severity	status	ref	location
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L41
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L47
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/new-vul.java#L49
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L10
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L19
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L28
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L33
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L35
		medium	open	master	vulns/very-new-vul.java#L41

Finding Title	Finding Description & Remediation	severity	status	ref	location
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.				
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L47
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L54
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/very-new-vul.java#L61
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/vuln.java#L10
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/vuln.java#L20
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/vuln.java#L30
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/vuln.java#L40
active-debug-code-getstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	vulns/vuln.java#L50

Finding Title	Finding Description & Remediation	severity	status	ref	location
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/Comment.java#L55
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/Comment.java#L70
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/Cowsay.java#L24
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/Postgres.java#L25
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/Postgres.java#L100
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/Postgres.java#L114
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/User.java#L34
active-debug-code-printstacktrace	Possible active debug code detected. Deploying an application with debug code can create unintended entry points or expose sensitive information.	medium	open	master	src/main/java/com/scalesec/vulnado/User.java#L58
	A cookie was detected without setting the 'secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure	medium	open	master	vulns/cookie-secure-flag.java#L23

Finding Title	Finding Description & Remediation	severity	status	ref	location
cookie-missing-secure-flag	channels such as HTTP. Set the 'secure' flag by calling 'cookie.setSecure(true);'				
use-of-md5	Detected MD5 hash algorithm which is considered insecure. MD5 is not collision resistant and is therefore not suitable as a cryptographic signature. Use HMAC instead.	medium	open	master	src/main/java/com/scalesec/vulnado/Postgres.java#L67
jdbc-sqli	Detected a formatted string in a SQL statement. This could lead to SQL injection if variables in the SQL statement are not properly sanitized. Use a prepared statements (java.sql.PreparedStatement) instead. You can obtain a PreparedStatement using 'connection.prepareStatement'.	medium	open	master	src/main/java/com/scalesec/vulnado/User.java#L49
cookie-secure-flag-false	A cookie was detected without setting the 'secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'secure' flag by calling 'cookie.setSecure(true);'	medium	open	master	vulns/cookie-secure-flag.java#L23
unrestricted-request-mapping	Detected a method annotated with 'RequestMapping' that does not specify the HTTP method. CSRF protections are not enabled for GET, HEAD, TRACE, or OPTIONS, and by default all HTTP methods are allowed when the HTTP method is not explicitly specified. This means that a method that performs state changes could be vulnerable to CSRF attacks. To mitigate, add the 'method' field and specify the HTTP method (such as 'RequestMethod.POST').	medium	open	master	src/main/java/com/scalesec/vulnado/CowController.java#L11
	Detected a method annotated with 'RequestMapping' that does not specify the HTTP	medium	open	master	src/main/java/com/scalesec/vulnado/LinksController.java#L15

Finding Title	Finding Description & Remediation	severity	status	ref	location
unrestricted-request-mapping	method. CSRF protections are not enabled for GET, HEAD, TRACE, or OPTIONS, and by default all HTTP methods are allowed when the HTTP method is not explicitly specified. This means that a method that performs state changes could be vulnerable to CSRF attacks. To mitigate, add the 'method' field and specify the HTTP method (such as 'RequestMethod.POST').				
unrestricted-request-mapping	Detected a method annotated with 'RequestMapping' that does not specify the HTTP method. CSRF protections are not enabled for GET, HEAD, TRACE, or OPTIONS, and by default all HTTP methods are allowed when the HTTP method is not explicitly specified. This means that a method that performs state changes could be vulnerable to CSRF attacks. To mitigate, add the 'method' field and specify the HTTP method (such as 'RequestMethod.POST').	medium	open	master	src/main/java/com/scalesec/vulnado/LinksController.java#L19
template-explicit-unescape	Detected an explicit unescape in a Mustache template, using triple braces '{{{...}}}' or ampersand '&'. 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	master	client/index.html#L73
hardcoded-salt	Cryptographic operations were identified that leverage a hardcoded salt/nonce. A salt does not need to remain secret, but should be random, generated from cryptographically secure sources of entropy, such as an CSPRNG. On iOS/macOS platforms, secure random data can be obtained via	medium	open	master	vulns/hardcoded_secrets.swift#L60

Finding Title	Finding Description & Remediation	severity	status	ref	location
	the `SecCopyRandomBytes` API available from RandomizationServices.				
hardcoded-salt	Cryptographic operations were identified that leverage a hardcoded salt/nonce. A salt does not need to remain secret, but should be random, generated from cryptographically secure sources of entropy, such as an CSPRNG. On iOS/macOS platforms, secure random data can be obtained via the `SecCopyRandomBytes` API available from RandomizationServices.	medium	open	master	vulns/hardcoded_secrets.swift#L74
hardcoded-symmetric-key	A hard-coded cryptographic key was detected. An attacker that obtains this key via reverse engineering or access to source code will be able to re-use this key to encrypt, decrypt, and/or sign data at will. Cryptographic keys should be unique, and randomly generated per user, per client.	medium	open	master	vulns/hardcoded_secrets.swift#L21
hardcoded-symmetric-key	A hard-coded cryptographic key was detected. An attacker that obtains this key via reverse engineering or access to source code will be able to re-use this key to encrypt, decrypt, and/or sign data at will. Cryptographic keys should be unique, and randomly generated per user, per client.	medium	open	master	vulns/hardcoded_secrets.swift#L57
insecure-crypto-aes-keysize	AES symmetric cryptographic operations were identified using a key size of 128bit which is less than the industry standard recommendation of 256bit.	medium	open	master	vulns/hardcoded_secrets.swift#L22
insecure-crypto-aes-keysize	AES symmetric cryptographic operations were identified using a key size of 128bit which is less than the industry standard recommendation of 256bit.	medium	open	master	vulns/hardcoded_secrets.swift#L40

Finding Title	Finding Description & Remediation	severity	status	ref	location
insecure-crypto-cbc-mode	Symmetric cryptographic operations were identified that use Cipher Block Chaining (CBC) mode. AES in CBC mode provides unauthenticated cryptographic encryption. CBC is also malleable, meaning that an attacker can influence the decrypted plaintext by modifying bits of the ciphertext (bit flipping attacks). Consider using an authenticated encryption mechanism, such as AES-GCM or ChaChaPoly. If CBC mode is **required**, consider augmenting the encryption with authentication by signing the ciphertexts with a Message Authentication Code (e.g. HMAC).	medium	open	master	vulns/hardcoded_secrets.swift#L16
insecure-crypto-cbc-mode	Symmetric cryptographic operations were identified that use Cipher Block Chaining (CBC) mode. AES in CBC mode provides unauthenticated cryptographic encryption. CBC is also malleable, meaning that an attacker can influence the decrypted plaintext by modifying bits of the ciphertext (bit flipping attacks). Consider using an authenticated encryption mechanism, such as AES-GCM or ChaChaPoly. If CBC mode is **required**, consider augmenting the encryption with authentication by signing the ciphertexts with a Message Authentication Code (e.g. HMAC).	medium	open	master	vulns/hardcoded_secrets.swift#L34
aws-subnet-has-public-ip-address	Resources in the AWS subnet are assigned a public IP address. Resources should not be exposed on the public internet, but should have access limited to consumers required for the function of your application. Set	medium	open	master	reverse_shell/main.tf#L33

Finding Title	Finding Description & Remediation	severity	status	ref	location
	`map_public_ip_on_launch` to false so that resources are not publicly-accessible.				
no-new-privileges	Service 'db' allows for privilege escalation via setuid or setgid binaries. Add 'no-new-privileges:true' in 'security_opt' to prevent this.	medium	open	master	docker-compose.yml#L23
writable-filesystem-service	Service 'db' 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	master	docker-compose.yml#L23

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/pro-engine-demo

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	9
Findings- SAST Medium Severity	18
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	main	src/main/java/hawk/api/jwt/JwtLog4jController.java#L24
tainted-sql-string	User data flows into this manually-constructed SQL string. User data can be safely inserted into SQL strings using prepared statements or an object-relational mapper (ORM). Manually-constructed SQL strings is a possible indicator of SQL injection, which could let an attacker steal or manipulate data from the database. Instead, use prepared statements (`connection.PreparedStatement`) or a safe library.	high	open	main	src/main/java/hawk/service/UserSearchService.java#L30
detected-private-key	Private Key detected. This is a sensitive credential and should not be hardcoded here. Instead, store this in a separate, private file.	high	open	main	src/main/resources/keyStore.pem#L5
spring-actuator-fully-enabled	Spring Boot Actuator is fully enabled. This exposes sensitive endpoints such as /actuator/env, /actuator/logfile, /actuator/heapdump and others. Unless you have Spring Security enabled or another means to protect these endpoints, this functionality is available without authentication, causing a significant security risk.	high	open	main	src/main/resources/application-postgresql.properties#L36
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/5/merge	src/assistant-fix-custom-message.java#L14
crlf-injection-logs-	When data from an untrusted source is put into a logger and not neutralized correctly, an attacker could	high	open		src/assistant-fix-custom-message.java#L14

Finding Title	Finding Description & Remediation	severity	status	ref	location
deepsemgrep-javaorg-copy	forge log entries or include malicious content. Please use the Jsoup.clean() function to sanitize data.			refs/pull/5/merge	
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	refs/pull/4/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/4/merge	src/assistant-fix-sqli-sequelize.ts#L5
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	open	main	src/main/java/hawk/api/jwt/JwtLog4jController.java#L24

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
cookie-missing-httponly	A cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie. Set the 'HttpOnly' flag by calling 'cookie.setHttpOnly(true);'	medium	open	main	src/main/java/hawk/controller/LoginController.java#L57
cookie-missing-secure-flag	A cookie was detected without setting the 'secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'secure' flag by calling 'new Cookie("XLOGINID", cookieCode).setSecure(true);'	medium	open	main	src/main/java/hawk/controller/LoginController.java#L57
cookie-missing-httponly	A cookie was detected without setting the 'HttpOnly' flag. The 'HttpOnly' flag for cookies instructs the browser to forbid client-side scripts from reading the cookie. Set the 'HttpOnly' flag by calling 'cookie.setHttpOnly(true);'	medium	open	main	src/main/java/hawk/controller/LoginController.java#L57
cookie-missing-samesite	The application does not appear to verify inbound requests which can lead to a Cross-site request forgery (CSRF) vulnerability. If the application uses cookie-based authentication, an attacker can trick users into sending authenticated HTTP requests without their knowledge from any arbitrary domain they visit. To prevent this vulnerability start by identifying if the framework or library leveraged has built-in features or offers plugins for CSRF protection. CSRF tokens should be unique and securely random. The \`Synchronizer Token\` or \`Double Submit Cookie\` patterns with defense-in-	medium	open	main	src/main/java/hawk/controller/LoginController.java#L57

Finding Title	Finding Description & Remediation	severity	status	ref	location
	depth mechanisms such as the `sameSite` cookie flag can help prevent CSRF. For more information, see: [Cross-site request forgery prevention](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html)				
cookie-missing-secure-flag	A cookie was detected without setting the 'secure' flag. The 'secure' flag for cookies prevents the client from transmitting the cookie over insecure channels such as HTTP. Set the 'secure' flag by calling 'new Cookie("XLOGINID", cookieCode).setSecure(true);'	medium	open	main	src/main/java/hawk/controller/LoginController.java#L57
spring-csrf-disabled	CSRF protection is disabled for this configuration. This is a security risk.	medium	open	main	src/main/java/hawk/MultiHttpSecurityConfig.java#L47
spring-csrf-disabled	CSRF protection is disabled for this configuration. This is a security risk.	medium	open	main	src/main/java/hawk/MultiHttpSecurityConfig.java#L88
spring-csrf-disabled	CSRF protection is disabled for this configuration. This is a security risk.	medium	open	main	src/main/java/hawk/MultiHttpSecurityConfig.java#L110
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/main/resources/templates/general.html#L30
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/main/resources/templates/login-form-multi.html#L15
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/main/resources/templates/login.html#L15
		medium	open	main	

Finding Title	Finding Description & Remediation	severity	status	ref	location
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.				src/main/resources/templates/search.html#L14
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks.	medium	open	main	src/main/resources/templates/user-search.html#L14
no-new-privileges	Service 'db' 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.yml#L3
no-new-privileges	Service 'javavulny' 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.yml#L12
writable-filesystem-service	Service 'db' 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.yml#L3
writable-filesystem-service	Service 'javavulny' 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.yml#L12
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/5/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: leaky-repo

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	18
Findings- SAST Medium Severity	5
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
detected-etc-shadow	linux shadow file detected	high	open	master	etc/shadow#L1
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L32
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L33
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L34
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L35
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L36
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L37
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L38
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L39
detected-bcrypt-hash	bcrypt hash detected	high	open	master	db/dump.sql#L40
	bcrypt hash detected	high	open	master	db/dump.sql#L41

Finding Title	Finding Description & Remediation	severity	status	ref	location
detected-bcrypt-hash					
detected-etc-shadow	linux shadow file detected	high	fixed	master	etc/shadow#L1
detected-generic-api-key	Generic API Key detected	high	open	master	.bashrc#L106
detected-generic-api-key	Generic API Key detected	high	open	master	cloud/.tugboat#L4
detected-private-key	Private Key detected. This is a sensitive credential and should not be hardcoded here. Instead, store this in a separate, private file.	high	open	master	.ssh/id_rsa#L1
detected-private-key	Private Key detected. This is a sensitive credential and should not be hardcoded here. Instead, store this in a separate, private file.	high	open	master	misc-keys/cert-key.pem#L1
detected-slack-token	Slack Token detected	high	open	master	.bash_profile#L23
dangerous-subprocess-use-audit	Detected subprocess function 'Popen' without a static string. If this data can be controlled by a malicious actor, it may be an instance of command injection. Audit the use of this call to ensure it is not controllable by an external resource. You may consider using 'shlex.escape()'.	high	open	master	.leaky-meta/benchmark.py#L27

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	master	.leaky-meta/benchmark.py#L16
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	master	.leaky-meta/benchmark.py#L45
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	master	.leaky-meta/benchmark.py#L161
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	master	.leaky-meta/benchmark.py#L163
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	master	.leaky-meta/benchmark.py#L165

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

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

Semgrep

Semgrep SAST Scan Report for Repository: Semgrep-Demo/just-another-service

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	0
Findings- SAST Medium Severity	0
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: bad-python-app

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	32
Findings- SAST Medium Severity	61
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	high	open	pre-commit-diff	templates/base.html#L18
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	high	open	pre-commit-diff	templates/base.html#L19
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in	high	open	pre-commit-diff	templates/base.html#L24

Finding Title	Finding Description & Remediation	severity	status	ref	location
	JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).				
var-in-script-tag	Detected a template variable used in a script tag. Although template variables are HTML escaped, HTML escaping does not always prevent cross-site scripting (XSS) attacks when used directly in JavaScript. If you need this data on the rendered page, consider placing it in the HTML portion (outside of a script tag). Alternatively, use a JavaScript-specific encoder, such as the one available in OWASP ESAPI. For Django, you may also consider using the 'json_script' template tag and retrieving the data in your script by using the element ID (e.g., `document.getElementById`).	high	open	pre-commit-diff	templates/base.html#L25
command-injection-os-system	Request data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the 'subprocess' module instead and pass the arguments as a list. See https://owasp.org/www-community/attacks/Command_Injection for more information.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L49
command-injection-os-system	Request data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the 'subprocess' module instead and pass the arguments as a list. See	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L54

Finding Title	Finding Description & Remediation	severity	status	ref	location
	https://owasp.org/www-community/attacks/Command_Injection for more information.				
subprocess-injection	Detected user input entering a `subprocess` call unsafely. This could result in a command injection vulnerability. An attacker could use this vulnerability to execute arbitrary commands on the host, which allows them to download malware, scan sensitive data, or run any command they wish on the server. Do not let users choose the command to run. In general, prefer to use Python API versions of system commands. If you must use subprocess, use a dictionary to allowlist a set of commands.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#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 the Django object-relational mappers (ORM) instead of raw SQL queries.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L21
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 the Django object-relational mappers (ORM) instead of raw SQL queries.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_search.py#L7

Finding Title	Finding Description & Remediation	severity	status	ref	location
os-system-injection	User data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the 'subprocess' module instead and pass the arguments as a list.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L15
os-system-injection	User data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the 'subprocess' module instead and pass the arguments as a list.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L21
os-system-injection	User data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the 'subprocess' module instead and pass the arguments as a list.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L27
dangerous-os-exec	Found user controlled content when spawning a process. This is dangerous because it allows a malicious actor to execute commands.	high	open	pre-commit-diff	vuln-1.py#L11
dangerous-subprocess-use	Detected subprocess function 'a' with user controlled data. A malicious actor could leverage this to perform command injection. You may consider using 'shlex.escape()'.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L36
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	reviewing	pre-commit-diff	vulns/file_upload/file_upload.py#L31
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier	high	open	pre-commit-diff	vulns/file_upload/file_upload.py#L49

Finding Title	Finding Description & Remediation	severity	status	ref	location
	to use without accidentally exposing a command injection vulnerability.				
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L15
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L21
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L27
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L50
dangerous-system-call	Found user-controlled data used in a system call. This could allow a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L55
		high	open		vuln-1.py#L11

Finding Title	Finding Description & Remediation	severity	status	ref	location
dangerous-os-exec-audit	Found dynamic content when spawning a process. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Ensure no external data reaches here.			pre-commit-diff	
dangerous-subprocess-use-audit	Detected subprocess function 'run' without a static string. If this data can be controlled by a malicious actor, it may be an instance of command injection. Audit the use of this call to ensure it is not controllable by an external resource. You may consider using 'shlex.escape()'.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L36
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/file_upload/file_upload.py#L31
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/file_upload/file_upload.py#L49
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L15

Finding Title	Finding Description & Remediation	severity	status	ref	location
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L21
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/semgrep_vulns.py#L27
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L50
dangerous-system-call-audit	Found dynamic content used in a system call. This is dangerous if external data can reach this function call because it allows a malicious actor to execute commands. Use the 'subprocess' module instead, which is easier to use without accidentally exposing a command injection vulnerability.	high	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L55
subprocess-shell-true	Found 'subprocess' function 'call' with 'shell=True'. This is dangerous because this call will spawn the command using a shell process. Doing so propagates current shell settings and variables, which makes it	high	open	pre-commit-diff	secretstest.py#L19

Finding Title	Finding Description & Remediation	severity	status	ref	location
	much easier for a malicious actor to execute commands. Use 'shell=False' instead.				
dangerous-subprocess-use-audit	Detected subprocess function 'call' without a static string. If this data can be controlled by a malicious actor, it may be an instance of command injection. Audit the use of this call to ensure it is not controllable by an external resource. You may consider using 'shlex.escape()'.	high	open	pre-commit-diff	secretstest.py#L19

Findings Summary- Medium Severity

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.	medium	open	pre-commit-diff	semgrep_sast_findings_bad-python-app_20231201-0103.html#L4
missing-integrity	This tag is missing an 'integrity' subresource integrity attribute. The 'integrity' attribute	medium	open	pre-commit-diff	semgrep_sast_findings_bad-python-app_20231201-0103.html#L247

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	medium	open	pre-commit-diff	semgrep_sast_findings_bad-python-app_20231201-0103.html#L753

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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	medium	open	pre-commit-diff	semgrep_sast_findings_bad-python-app_20231201-0103.html#L783

Finding Title	Finding Description & Remediation	severity	status	ref	location
	(file) youâ€™re telling the browser to fetch in the 'integrity' attribute for all externally hosted files.				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main-10.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main-10.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main-2.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main-2.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main-3.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main-3.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main-4.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main-4.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main-7.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main-7.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main-9.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main-9.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
java-jwt-hardcoded-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	pre-commit-diff	vuln-main.java#L15
java-jwt-hardcoded-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	medium	open	pre-commit-diff	vuln-main.java#L46

Finding Title	Finding Description & Remediation	severity	status	ref	location
	securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L16
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal	medium	open	pre-commit-diff	templates/components/navbar.html#L26

Finding Title	Finding Description & Remediation	severity	status	ref	location
	forward slash and concatenate the URL, like this: href='/ {{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/ {{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L29
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks.	medium	open	pre-commit-diff	templates/components/navbar.html#L41

Finding Title	Finding Description & Remediation	severity	status	ref	location
	If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L44
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and	medium	open	pre-commit-diff	templates/components/navbar.html#L54

Finding Title	Finding Description & Remediation	severity	status	ref	location
	is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L58
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a	medium	open	pre-commit-diff	templates/components/navbar.html#L62

Finding Title	Finding Description & Remediation	severity	status	ref	location
	malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.				
var-in-href	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L66
var-in-href	Detected a template variable used in an	medium	open		templates/components/navbar.html#L70

Finding Title	Finding Description & Remediation	severity	status	ref	location
	anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. If using a relative URL, start with a literal forward slash and concatenate the URL, like this: href='/{{link}}'. You may also consider setting the Content Security Policy (CSP) header.			pre-commit-diff	
template-autoescape-off	Detected a template block where autoescaping is explicitly disabled with '{% autoescape off %}'. This allows rendering of raw HTML in this segment. Turn autoescaping on to prevent cross-site scripting (XSS). If you must do this, consider instead, using `mark_safe` in Python code.	medium	open	pre-commit-diff	semgrep_sast_findings_bad-python-app_20231201-0103.html#L458
template-autoescape-off	Detected a template block where autoescaping is explicitly	medium	open	pre-commit-diff	semgrep_sast_findings_bad-python-app_20231201-0103.html#L479

Finding Title	Finding Description & Remediation	severity	status	ref	location
	disabled with '{% autoescape off %}'. This allows rendering of raw HTML in this segment. Turn autoescaping on to prevent cross-site scripting (XSS). If you must do this, consider instead, using `mark_safe` in Python code.				
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L16
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site	medium	open	pre-commit-diff	templates/components/navbar.html#L26

Finding Title	Finding Description & Remediation	severity	status	ref	location
	scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.				
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L44
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also	medium	open	pre-commit-diff	templates/components/navbar.html#L54

Finding Title	Finding Description & Remediation	severity	status	ref	location
	consider setting the Content Security Policy (CSP) header.				
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L58
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L62

Finding Title	**Finding Description & Remediation**	**severity**	**status**	**ref**	**location**
template-href-var	Detected a template variable used in an anchor tag with the 'href' attribute. This allows a malicious actor to input the 'javascript:' URI and is subject to cross- site scripting (XSS) attacks. Use the 'url' template tag to safely generate a URL. You may also consider setting the Content Security Policy (CSP) header.	medium	open	pre-commit-diff	templates/components/navbar.html#L66
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks	medium	open	pre-commit-diff	templates/file_upload.html#L5
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks	medium	open	pre-commit-diff	templates/idor/idor_login.html#L15
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks	medium	open	pre-commit-diff	templates/ssrf.html#L9

Finding Title	**Finding Description & Remediation**	**severity**	**status**	**ref**	**location**
django-no-csrf-token	Manually-created forms in django templates should specify a csrf_token to prevent CSRF attacks	medium	open	pre-commit-diff	templates/xss-stored.html#L10
user-eval	Found user data in a call to 'eval'. This is extremely dangerous because it can enable an attacker to execute arbitrary remote code on the system. Instead, refactor your code to not use 'eval' and instead use a safe library for the specific functionality you need.	medium	open	pre-commit-diff	vulns/semgrep_vulns.py#L31
avoid_app_run_with_bad_host	Running flask app with host 0.0.0.0 could expose the server publicly.	medium	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L56
debug-enabled	Detected Flask app with debug=True. Do not deploy to production with this flag enabled as it will leak sensitive information. Instead, consider using Flask configuration variables or setting 'debug' using	medium	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L56

Finding Title	Finding Description & Remediation	severity	status	ref	location
	system environment variables.				
render-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	medium	open	pre-commit-diff	middlewares.py#L16
secure-set-cookie	Found a Flask cookie with insecurely configured properties. By default the secure, httponly and samesite ar configured insecurely. cookies should be handled securely by setting `secure=True`, `httponly=True`, and `samesite='Lax'` in response.set_cookie(...). If these parameters are not properly set, your cookies are not properly protected and are at risk of being stolen by an attacker. Include the `secure=True`, `httponly=True`, `samesite='Lax'` arguments or set these to be true in the Flask configuration.	medium	open	pre-commit-diff	vulns/idor/idor.py#L33

Finding Title	Finding Description & Remediation	severity	status	ref	location
secure-set-cookie	Found a Flask cookie with insecurely configured properties. By default the secure, httponly and samesite ar configured insecurely. cookies should be handled securely by setting \`secure=True\`, \`httponly=True\`, and \`samesite='Lax'\` in response.set_cookie(...). If these parameters are not properly set, your cookies are not properly protected and are at risk of being stolen by an attacker. Include the \`secure=True\`, \`httponly=True\`, \`samesite='Lax'\` arguments or set these to be true in the Flask configuration.	medium	open	pre-commit-diff	vulns/idor/idor.py#L34
template-autoescape-off	Detected a segment of a Flask template where autoescaping is explicitly disabled with '{% autoescape off %}'. This allows rendering of raw HTML in this segment. Ensure no user data is rendered here, otherwise	medium	open	pre-commit-diff	templates/xss-reflected.html#L13

Finding Title	Finding Description & Remediation	severity	status	ref	location
	this is a cross-site scripting (XSS) vulnerability, or turn autoescape on.				
template-autoescape-off	Detected a segment of a Flask template where autoescaping is explicitly disabled with '{% autoescape off %}'. This allows rendering of raw HTML in this segment. Ensure no user data is rendered here, otherwise this is a cross-site scripting (XSS) vulnerability, or turn autoescape on.	medium	open	pre-commit-diff	templates/xss-stored.html#L29
dynamic-urllib-use-detected	Detected a dynamic value being used with urllib. urllib supports 'file://' schemes, so a dynamic value controlled by a malicious actor may allow them to read arbitrary files. Audit uses of urllib calls to ensure user data cannot control the URLs, or consider using the 'requests' library instead.	medium	open	pre-commit-diff	vulns/ssrf/ssrf.py#L35
eval-detected		medium	open		vulns/semgrep_vulns.py#L31

Finding Title	Finding Description & Remediation	severity	status	ref	location
	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.			pre-commit-diff	
md5-used-as-password	It looks like MD5 is used as a password hash. MD5 is not considered a secure password hash because it can be cracked by an attacker in a short amount of time. Use a suitable password hashing function such as scrypt. You can use `hashlib.scrypt`.	medium	open	pre-commit-diff	vulns/idor/idor.py#L14
md5-used-as-password	It looks like MD5 is used as a password hash. MD5 is not considered a secure password hash because it can be cracked by an attacker in a short amount of time. Use a suitable password hashing function such as	medium	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L19

Finding Title	Finding Description & Remediation	severity	status	ref	location
	scrypt. You can use `hashlib.scrypt`.				
md5-used-as-password	It looks like MD5 is used as a password hash. MD5 is not considered a secure password hash because it can be cracked by an attacker in a short amount of time. Use a suitable password hashing function such as scrypt. You can use `hashlib.scrypt`.	medium	open	pre-commit-diff	vulns/sql_injection/sql_injection_login.py#L44
flask-duplicate-handler-name	Looks like `route_param_concat` is a flask function handler that registered to two different routes. This will cause a runtime error	medium	open	pre-commit-diff	vulns/semgrep_vulns.py#L17
pass-body-fn	`pass` is the body of function before_request. Consider removing this or raise NotImplementedError() if this is a TODO	medium	open	pre-commit-diff	app.py#L30
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting	medium	open	pre-commit-diff	semgrep_sast_findings_report_sh.py#L237

Finding Title	Finding Description & Remediation	severity	status	ref	location
	files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").				
unspecified-open-encoding	Missing 'encoding' parameter. 'open()' uses device locale encodings by default, corrupting files with special characters. Specify the encoding to ensure cross-platform support when opening files in text mode (e.g. encoding="utf-8").	medium	open	pre-commit-diff	semgrep_sast_findings_report_sh.py#L264
is-function-without-parentheses	Is "is_admin" a function or an attribute? If it is a function, you may have meant self.is_admin() because self.is_admin is always true.	medium	open	pre-commit-diff	db_models.py#L6
unchecked-subprocess-call	This is not checking the return value of this subprocess call; if it fails no exception will be raised. Consider subprocess.check_call() instead	medium	open	pre-commit-diff	secretstest.py#L19

Finding Title	Finding Description & Remediation	severity	status	ref	location
return-not-in-function	`return` only makes sense inside a function	medium	open	pre-commit-diff	secretstest.py#L20

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: securingsoftware/ forum-service

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	4
Findings- SAST Medium Severity	0
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	update_products	sqli-sequelize.ts#L5
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	update_products	sqli-sequelize.ts#L5
deno-dangerous-run	Detected non-literal calls to Deno.run(). This could lead to a command injection vulnerability.	high	open	main	src/deno-dangerous-run.js#L12
shelljs-exec-injection	If unverified user data can reach the `exec` method it can result in Remote Code Execution	high	open	main	src/shelljs-exec-injection.js#L5

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/CodeSnippets

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	4
Findings- SAST Medium Severity	1
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/3/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	open	refs/pull/3/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 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	refs/pull/2/merge	src/assistant-fix-sqli-sequelize.ts#L5
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/2/merge	src/assistant-fix-sqli-sequelize.ts#L5

Findings Summary- Medium Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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/3/merge	src/assistant-fix-custom-message.java#L13

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location

Semgrep SAST Scan Report for Repository: Semgrep-Demo/sharpcompress

Report Generated at 2024-09-04 21:52

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	4
Findings- SAST Medium Severity	9
Findings- SAST Low Severity	0

Findings Summary- High Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location
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	open	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 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	refs/pull/1/merge	src/assistant-fix-sqli-sequelize.ts#L5
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
narrow-to-wide-string-mismatch	A byte-string (narrow string) is used in an API that expects a wide-string. This can trigger an out-of-bounds read.	medium	open	master	reference/unrar/pathfn.cpp#L165
narrow-to-wide-string-mismatch	A byte-string (narrow string) is used in an API that expects a wide-string. This can trigger an out-of-bounds read.	medium	open	master	reference/unrar/pathfn.cpp#L167
narrow-to-wide-string-mismatch	A byte-string (narrow string) is used in an API that expects a wide-string. This can trigger an out-of-bounds read.	medium	open	master	reference/unrar/pathfn.cpp#L864
narrow-to-wide-string-mismatch	A byte-string (narrow string) is used in an API that expects a wide-string. This can trigger an out-of-bounds read.	medium	open	master	reference/unrar/strfn.cpp#L320
narrow-to-wide-string-mismatch	A byte-string (narrow string) is used in an API that expects a wide-string. This can trigger an out-of-bounds read.	medium	open	master	reference/unrar/strfn.cpp#L323
tainted-allocation-size	Externally controlled data influences the size of an allocation. This can usually lead to overflow or underflow and later trigger an out of bounds conditions.	medium	open	master	reference/unrar/cmddata.cpp#L39
wide-to-narrow-string-mismatch	A wide-string is used in an API that should consume byte-string (narrow string). This can trigger an out-of-bounds read.	medium	open	master	reference/unrar/pathfn.cpp#L790
world-writable-file	This call makes a world-writable file which allows any user on a machine to write to the file. This may allow	medium	open	master	reference/unrar/file.cpp#L192

Finding Title	Finding Description & Remediation	severity	status	ref	location
	attackers to influence the behaviour of this process by writing to the file.				
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

Findings Summary- Low Severity

Finding Title	Finding Description & Remediation	severity	status	ref	location