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.				