Semgrep SAST Scan Report for Repository: Nitin-Vulnerable-Flask-App

Report Generated at 2024-02-28 09:28

SAST Scan Summary

Vulnerability Severity	Vulnerability Count
Findings- SAST High Severity	11
Findings- SAST Medium Severity	17
Findings- SAST Low Severity	0

Findings Summary- HIGH Severity

Finding Title	Finding Description & Remediation	severity	state	ref	location
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	unresolved	master	app/static/loader.js#L153
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	unresolved	master	app/static/loader.js#L153
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	unresolved	master	app/app.py#L261
tainted-pyyaml-flask	Insecure deserialization (called pickling in python) is when user-controllable data is deserialized by an application. This potentially enables an attacker to manipulate serialized objects in order to pass harmful data into the application code and may result in arbitrary code execution, OS command injection or DoS. Many deserialization-based attacks are completed before deserialization is finished. This means that the deserialization process itself can initiate an attack, even if the app's own functionality does not directly interact with the malicious object. PyYAML's `yaml` module is as powerful as `pickle` and so may call any Python function. It is recommended to secure your application by using `yaml.SafeLoader` or `yaml.CSafeLoader`.	high	unresolved	master	app/app.py#L329
dangerous-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	high	unresolved	master	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	unresolved	master	app/app.py#L271

Finding Title	Finding Description & Remediation	severity	state	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 SQLAlchemy which will protect your queries.	high	unresolved	master	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	unresolved	master	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	unresolved	master	app/app.py#L184
unverified-jwt-decode	Detected JWT token decoded with 'verify=False'. This bypasses any integrity checks for the token which means the token could be tampered with by malicious actors. Ensure that the JWT token is verified.	high	unresolved	master	app/app.py#L97
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 Definition Language. In most cases, SQLAlchemy ORM will be a better option.	high	unresolved	master	app/app.py#L265

Findings Summary- MEDIUM Severity

Finding Title	Finding Description & Remediation	severity	state	ref	location
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	unresolved	master	app/static/loader.js#L24
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	unresolved	master	app/static/loader.js#L26
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	unresolved	master	app/static/loader.js#L41
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	unresolved	master	app/templates/index.html#L12
detect-non-literal-regexp	RegExp() called with a \`c\` function argument, this might allow an attacker to cause a Denial of Service (DoS) within your application as RegExP which blocks the main thread.	medium	unresolved	master	app/static/loader.js#L55
detect-non-literal-regexp	RegExp() called with a \`c\` function argument, this might allow an attacker to cause a Denial of Service (DoS) within your application as RegExP which blocks the main thread.	medium	unresolved	master	app/static/loader.js#L55
detect-non-literal-regexp	RegExp() called with a \`c\` function argument, this might allow an attacker to cause a Denial of Service (DoS) within your application as RegExP which blocks the main thread.	medium	unresolved	master	app/static/loader.js#L59

Finding Title	Finding Description & Remediation	severity	state	ref	location
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	unresolved	master	app/static/loader.js#L3
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	unresolved	master	app/static/loader.js#L106
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	unresolved	master	app/templates/index.html#L12
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	unresolved	master	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	unresolved	master	app/app.py#L114

Finding Title	**Finding Description & Remediation**	**severity**	**state**	**ref**	**location**
render-template-string	Found a template created with string formatting. This is susceptible to server-side template injection and cross-site scripting attacks.	medium	unresolved	master	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	unresolved	master	app/app.py#L103
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	unresolved	master	app/templates/index.html#L12
formatted-sql-query	Detected possible formatted SQL query. Use parameterized queries instead.	medium	unresolved	master	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 an attacker in a short amount of time. Use a suitable password hashing function such as scrypt. You can use `hashlib.scrypt`.	medium	unresolved	master	app/app.py#L141

Findings Summary- LOW Severity

Finding Title	Finding Description & Remediation	severity	state	ref	location