Daksh SCRA – Source Code Analysis Report

Dec 20, 2024

Scan Summary

This section provides a summary of the selected inputs and essential metrics collected during the scanning process. It offers an overview of the key information, allowing users to grasp the important aspects of the scan at a glance. For more detailed information about the identified areas of interest, please refer to the respective sections.

```
[+] Inputs Selected:
    [-] Target Directory: sample/sample_programs/vulnerable_py_app/
    [-] Rule Selected: python
    [-] Total Rules Loaded: 54
        [-] Platform Specific Rules: python[30]
        [-] Common Rules: 24
    [-] File Types Selected: python
[+] Detection Summary:
    [-] Total Project Files Identified: 5
    [-] Total Files Identified (Based on Selected Rule): 1
    [-] Total Files Scanned (Based on Selected Rule): 1
    [-] File Extensions Identified (Based on Selected Rule):
        python: [.py]
    [-] Code Files - Areas-of-Interests (Rules Matched): 7
    [-] File Paths - Areas-of-Interests (Rules Matched): 0

[+] Scanning Timeline:
    [-] Scan start time: 2024-12-20 12:15:49
    [-] Scan end time: 2024-12-20 12:15:49
    [-] Scan completed in: 00Hr:00Min:00s:043ms
```

Security - Areas of Interest

This section lists the key areas within the source code that need to be examined for identifying potential security weaknesses.

The code reviewer should carefully examine the identified areas and review any reported code snippets and file paths to validate the presence of any potential vulnerabilities. The validation process should involve a thorough analysis of the code and its associated components to determine the extent of the potential security risk. Any issues identified during the validation process should be documented.

PYTHON FINDINGS

ID: PYTHON-1 : SQL Query: SELECT Statements with WHERE or ORDER BY

Rule Description : Detects instances of SELECT statements with WHERE or ORDER BY clauses in SQL queries, which may be vulnerable to SQL injection if user input is not handled securely.

Issue Description : If this rule matches, it indicates the use of potentially vulnerable SELECT statements with WHERE or ORDER BY clauses. Improper handling of user input in these clauses can lead to SQL injection, allowing attackers to manipulate query behavior and gain unauthorized access to data.

Developer Note : Developers should validate and implement secure alternatives by using parameterized queries with placeholders or bindings to prevent SQL injection risks.

Reviewer Note : Reviewers should check for insecure usage of SELECT statements with WHERE or ORDER BY clauses, verifying the use of parameterized queries and input sanitization to prevent SQL injection.

- Source File : vulnerable_py_app/app.py

```
1 [26]     query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
```

ID: PYTHON-2 : SQL Query: Unrestricted SELECT *

Rule Description : Detects "SELECT * FROM" in SQL queries, which can expose unnecessary or sensitive data.

Issue Description : If this rule matches, it suggests that the query retrieves more data than necessary, potentially exposing sensitive information and increasing the attack surface.

Developer Note : Developers should validate existing queries and specify only the required columns to prevent unnecessary data exposure.

Reviewer Note : Reviewers should check for insecure implementation using "SELECT *" and ensure only essential columns are specified to limit data exposure risks.

- Source File : vulnerable_py_app/app.py

```
1 [26]     query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
```

COMMON FINDINGS

ID: COMMON-1 : Authentication Modules

Rule Description : Detects common terms used in authentication functionalities, such as 'Login', 'authenticate', 'OAuth', and 'JWT', when used in likely function names or module references.

Issue Description : This rule matches on common function or module names associated with authentication. If not properly secured, these modules may expose vulnerabilities in user authentication, allowing unauthorized access.

Developer Note : Ensure authentication processes use secure configurations, apply best practices, and avoid using weak or generic names that could be predictable.

Reviewer Note : Verify that authentication mechanisms are implemented securely, with protections like parameterized inputs, encryption, and secure session management.

- Source File : vulnerable_py_app/app.py

```
[22] @app.route("/login", methods=["POST"])
[23] def login():
```

ID: COMMON-2 : File Upload Functionality

Rule Description : Detects likely implementations of file upload functionality by matching common terms associated with file upload services or modules.

Issue Description : If this rule matches, it indicates potential file upload functionality, which, if improperly secured, could allow unauthorized or malicious file uploads. This can lead to risks such as arbitrary code execution or data leakage.

Developer Note : Ensure file upload functionality is implemented securely with validation of file types, size limits, and access controls. Use established libraries or frameworks with built-in security features for file uploads to minimize risks.

Reviewer Note : Verify that secure file upload practices are followed, including file type validation, size restrictions, and appropriate access controls to prevent unauthorized file uploads.

- Source File : vulnerable_py_app/app.py

```
[36] # Insecure File Upload
[42]         return "File uploaded!"
```

ID: COMMON-3 : Password

Rule Description : Detects potential password-related strings in the code.

Issue Description : If this rule matches, it indicates the potential presence of password-related strings in the code, which can lead to security risks if not handled properly.

Developer Note : Developers should follow best practices for password handling, including strong encryption, salted hashing, and enforcing secure password policies.

Reviewer Note : Reviewers should assess the password handling mechanisms and verify if proper security measures are in place.

- Source File : vulnerable_py_app/app.py

```
[13]     cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, username TEXT, password TEXT)")
[25]     password = request.form.get("password")
```

ID: COMMON-4 : Sensitive Debug Information

Rule Description : Detects the presence of sensitive debug information in application settings.

Issue Description : If this rule matches, it indicates a potential security concern related to the presence of sensitive debug information in production settings. Enabling debugging in production can expose sensitive information and pose security risks. Developers should ensure that debugging is disabled in production environments.

Developer Note : Developers should disable debugging features or set debug flags to false in application settings for production environments to prevent the exposure of sensitive debug information.

Reviewer Note : Reviewers should check for debug-related configurations in application settings and ensure that they are disabled or set to false in production environments to protect sensitive information.

- Source File : vulnerable_py_app/app.py

```
[47]     app.run(debug=True)
```

ID: COMMON-5 : Unvalidated Redirects and Forwards

Rule Description : Detects potential unvalidated redirect and forward vulnerabilities.

Issue Description : If this rule matches, it indicates the presence of code patterns that may indicate vulnerabilities in unvalidated redirects and forwards, which can be exploited by attackers to redirect users to malicious websites or perform phishing attacks.

Developer Note : Developers should validate and sanitize all user-supplied input used in redirect and forward operations. They should also ensure that redirects and forwards are performed only to trusted and authorized destinations.

Reviewer Note : Reviewers should review the code for potential unvalidated redirect and forward vulnerabilities and verify if proper input validation and destination checks are implemented.

Parsed Paths – Areas of Interest

This section contains a list of file paths that have been identified by matching them with a predefined set of keywords. These files are typically of interest to a code reviewer, and should be examined for possible security vulnerabilities or insecure implementations.

Identified Files Path

In this section, you'll find a comprehensive list of project file paths that were parsed during the analysis. This list was generated based on the selected file types during the scanning process and was used to identify the areas of interest listed in the previous sections. It's important to note that this list includes all the files, making it a superset of the list provided in the "Parsed Paths - Areas of Interest" section.

It is still recommended for code reviewers to review this list for any potentially interesting files that may have been overlooked in the "Parsed Paths - Areas of Interest" section.

- Source File : vulnerable_py_app/app.py