MyESI Automated Compliance & Vulnerability Assessment

Project: myesi-user-service

Generated: 2025-11-16 05:38 UTC **User:** 1

Note: Compliance score unavailable: Limit reached. Next reset at 2025-11-17 00:00:00+00

1. Executive Summary

• Compliance Score: **0%**

• Average Risk Score: **0.00**

• Total Vulnerabilities: **0**

Standards Compliance:

Standard	Status
ISO_27001	Not Available (subscription limit reached)
NIST_SP_800_53	Not Available (subscription limit reached)
OWASP	Not Available (subscription limit reached)

2. Compliance Controls Overview

No compliance controls available (possibly due to SBOM limit reached).

3. Vulnerability Findings

No vulnerabilities available (SBOM missing or quota exceeded).

4. Code Findings (Static Analysis)

1. python.fastapi.web.fastapi-cookie-samesite-none.fastapi-cookie-samesite-none (Info / HIGH)

/app/tmp/repos/semgrep-20251116053804/app/api/v1/users.py

Detected a cookie options with the `SameSite` flag set to "None". This is a potential security risk that arises from the way web browsers manage cookies. In a typical web application, cookies are used to store and transmit session-related data between a client and a server. To enhance security, cookies can be marked with the "SameSite" attribute, which restricts their usage based on the origin of the page that set them. This attribute can have three values: "Strict," "Lax," or "None". Make sure that the choice of the `None` value is intentional and that you understand the potential security implications. In FastAPI apps, the `set_cookie` function's argument `samesite` is set to 'Lax' by default. While 'Strict' is the most secure option, 'Lax' is a good compromise between security and usability and this default value is secure for most applications. Do not set `samesite` to 'None' to turn off this security feature.

samesite="none",

https://owasp.org/Top10/A01_2021-Broken_Access_Control
https://web.dev/articles/samesite-cookies-explained
https://www.starlette.io/responses/

Suggested Fix:

Summary The FastAPI framework may be improperly setting the SameSite attribute for cookies, potentially allowing cross-site request forgery (CSRF) attacks. This configuration could expose user sessions to unauthorized access if cookies are sent with cross-origin requests. Remediation Steps Set the SameSite Attribute for Cookies Ensure that cookies are configured with the SameSite attribute to mitigate CSRF risks. Example Code Adjustment Modify your FastAPI cookie settings as follows: ```python from fastapi import FastAPI, Response app = FastAPI() @app.get("/") def read_root(response: Response): response.set_cookie(key="my_cookie", value="cookie_value", samesite="Strict") return {"message": "Cookie set with SameSite attribute"} ``` Configuration Guidance Review your FastAPI application to ensure that all cookies set by the application include the SameSite attribute. Depending on your application's needs, you may choose Strict, Lax, or None (with secure context) for the SameSite value. Always prefer Strict or Lax unless cross-site usage is explicitly required.

2. python.django.web.django-cookie-samesite-none.django-cookie-samesite-none (Info / HIGH)

/app/tmp/repos/semgrep-20251116053804/app/api/v1/users.py

Detected a cookie options with the `SameSite` flag set to "None". This is a potential security risk that arises from the way web browsers manage cookies. In a typical web application, cookies are used to store and transmit session-related data between a client and a server. To enhance security, cookies can be marked with the "SameSite" attribute, which restricts their usage based on the origin of the page that set them. This attribute can have three values: "Strict," "Lax," or "None". Make sure the `SameSite` attribute of the important cookies (e.g., session cookie) is set to a reasonable value. When `SameSite` is set to "Strict", no 3rd party cookie will be sent with outgoing requests, this is the most secure and private setting but harder to deploy with good usability. Setting it to "Lax" is the minimum requirement.

samesite="none",

https://owasp.org/Top10/A01_2021-Broken_Access_Control
https://web.dev/articles/samesite-cookies-explained

Suggested Fix:

Summary The django-cookie-samesite-none vulnerability indicates that cookies in your Django application are not utilizing the SameSite attribute effectively. This can lead to CSRF (Cross-Site Request Forgery) vulnerabilities, as cookies are sent with cross-site requests. Proper configuration helps mitigate potential security risks. Remediation Steps Update Cookie Settings: Ensure that cookies are set with the SameSite attribute to restrict how cookies are sent with cross-origin requests. Example Configuration: python # In your Django settings.py CSRF_COOKIE_SAMESITE = 'Lax' # or 'Strict' based on your requirements SESSION_COOKIE_SAMESITE = 'Lax' # or 'Strict' Review Cookie Policy: Evaluate your application's cookie policy to determine if cookies need to be accessible in a cross-site context. Set SameSite=None only when absolutely necessary and ensure Secure is also set for those cookies.

3. python.fastapi.web.fastapi-cookie-samesite-none.fastapi-cookie-samesite-none (Info / HIGH)

/app/tmp/repos/semgrep-20251116051159/app/api/v1/users.py

Detected a cookie options with the `SameSite` flag set to "None". This is a potential security risk that arises from the way web browsers manage cookies. In a typical web application, cookies are used to store and transmit session-related data between a client and a server. To enhance security, cookies can be marked with the "SameSite" attribute, which restricts their usage based on the origin of the page that set them. This attribute can have three values: "Strict," "Lax," or "None". Make sure that the choice of the `None` value is intentional and that you understand the potential security implications. In FastAPI apps, the `set_cookie` function's argument `samesite` is set to 'Lax' by default. While 'Strict' is the most secure option, 'Lax' is a good compromise between security and usability and this default value is secure for most applications. Do not set `samesite` to 'None' to turn off this security feature.

```
samesite="none",
```

https://owasp.org/Top10/A01_2021-Broken_Access_Control
https://web.dev/articles/samesite-cookies-explained
https://www.starlette.io/responses/

Suggested Fix:

Summary The fastapi-cookie-samesite-none vulnerability indicates that cookies are being sent with the SameSite=None attribute without the Secure flag. This can lead to Cross-Site Request Forgery (CSRF) attacks as cookies may be sent in cross-origin requests. Remediation Steps Update Cookie Configuration: Ensure that all cookies that have SameSite=None are also marked as Secure. Example Code Update: ```python from fastapi import FastAPI, Response app = FastAPI() @app.get("/set-cookie") def set_cookie(response: Response): response.set_cookie(key="my_cookie", value="cookie_value", samesite="None", secure=True) return {"message": "Cookie set with SameSite=None and Secure flag"} `` 3. **Policy Guidance**: Review and update your cookie management policies to ensure that any cookie usingSameSite=Noneis always accompanied by theSecure` attribute to enhance security against CSRF vulnerabilities.

4. python.django.web.django-cookie-samesite-none.django-cookie-samesite-none (Info / HIGH)

/app/tmp/repos/semgrep-20251116051159/app/api/v1/users.py

Detected a cookie options with the `SameSite` flag set to "None". This is a potential security risk that arises from the way web browsers manage cookies. In a typical web application, cookies are used to store and transmit session-related data between a client and a server. To enhance security, cookies can be marked with the "SameSite" attribute, which restricts their usage based on the origin of the page that set them. This attribute can have three values: "Strict," "Lax," or "None". Make sure the `SameSite` attribute of the important cookies (e.g., session cookie) is set to a reasonable value. When `SameSite` is set to "Strict", no 3rd party cookie will be sent with outgoing requests, this is the most secure and private setting but harder to deploy with good usability. Setting it to "Lax" is the minimum requirement.

```
samesite="none",
```

https://owasp.org/Top10/A01_2021-Broken_Access_Control
https://web.dev/articles/samesite-cookies-explained

Suggested Fix:

Summary The django-cookie-samesite-none vulnerability indicates that cookies may not have the SameSite attribute set correctly, which can lead to cross-site request forgery (CSRF) attacks. Setting the SameSite attribute helps mitigate the risk of unwanted cross-origin requests. Remediation Steps Update Cookie Settings: Ensure that all cookies set by your Django application include the SameSite attribute. Example Code Update: In your Django settings, modify your cookie settings as follows: python SESSION_COOKIE_SAMESITE = 'Lax' # or 'Strict', based on your needs CSRF_COOKIE_SAMESITE = 'Lax' # or 'Strict', based on your needs Review Cookie Policy: Assess your application’s cookie usage and determine the appropriate SameSite value (Lax, Strict, or None). For cookies that require cross-origin requests, use SameSite=None; Secure to ensure they are sent over HTTPS only.

5. dockerfile.security.missing-user.missing-user (Error / MEDIUM)

/app/tmp/repos/semgrep-20251116053804/Dockerfile

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'.

```
CMD uvicorn app.main:app --host 0.0.0.0 --port 8001 --workers 1 --reload
```

https://owasp.org/Top10/A04_2021-Insecure_Design

Suggested Fix:

Summary The Dockerfile is configured to run as the root user, which poses a security risk by increasing the attack surface and potential impact of a compromised container. Running containers as non-root users minimizes the risk of privilege escalation and limits the damage that can be done if the container is compromised. Remediation Steps Create a non-root user in your Dockerfile to run your application. Example command to add a user and switch to that user: dockerfile RUN adduser --disabled-password --gecos '' myuser USER myuser Policy Guidance: Always define a non-root user in your Dockerfile, and ensure that any necessary permissions are granted explicitly to that user. Avoid using the root user unless absolutely necessary.

6. dockerfile.security.missing-user.missing-user (Error / MEDIUM)

/app/tmp/repos/semgrep-20251116051159/Dockerfile

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'.

CMD uvicorn app.main:app --host 0.0.0.0 --port 8001 --workers 1 --reload

https://owasp.org/Top10/A04_2021-Insecure_Design

Suggested Fix:

Summary The vulnerability identified indicates that the Dockerfile does not specify a user to run the application, which can lead to security risks such as privilege escalation and unauthorized access to host resources. Remediation Steps Specify a Non-Root User: Modify the Dockerfile to create and specify a non-root user for running the application. Example Command: Dockerfile RUN addgroup --system myusergroup && adduser --system --ingroup myusergroup myuser USER myuser Configuration Guidance: Ensure that the application has the necessary permissions to run under the non-root user. Review all file permissions and ownerships accordingly to avoid runtime issues. Consider using USER directive in your Dockerfile to enforce the least privilege principle.