MyESI Automated Compliance & Vulnerability Assessment

Project: myesi-user-service

Generated: 2025-11-16 05:44 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-cookie-samesite-none vulnerability indicates that cookies may be set without the SameSite attribute, which can expose applications to Cross-Site Request Forgery (CSRF) attacks. Ensuring proper cookie configurations is crucial for maintaining session integrity and protecting user data. Remediation Steps Update Cookie Settings: Ensure that all cookies set by your FastAPI application include the SameSite attribute. Example Code: Update your cookie setting in FastAPI like this: ```python from fastapi import FastAPI, Response app = FastAPI() @app.get("/set-cookie") async def set_cookie(response: Response): response.set_cookie(key="my_cookie", value="cookie_value", samesite="Lax") return {"message": "Cookie set with SameSite=Lax"} `` 3. **Policy Guidance**: Review all cookie configurations in your application and ensure that they are set toSameSite=LaxorSameSite=Strictas appropriate. Avoid usingSameSite=Noneunless absolutely necessary and ensure that theSecure` flag is also set for those cookies.

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 use of the SameSite=None attribute for cookies in Django can expose applications to Cross-Site Request Forgery (CSRF) attacks. It allows cookies to be sent with requests originating from different sites, which can lead to unauthorized actions being performed on behalf of authenticated users. Remediation Steps Review Cookie Usage: Assess all cookies that are set with SameSite=None to determine if they truly require cross-site access. Set Appropriate SameSite Attribute: Change the SameSite attribute of cookies to Lax or Strict where applicable. - Example: response.set_cookie('cookie_name', 'cookie_value', samesite='Lax') Update Django Settings: Ensure your Django settings are configured to set the SameSite attribute appropriately for all cookies. - Configuration: In your settings.py, set: python CSRF_COOKIE_SAMESITE = 'Lax' SESSION_COOKIE_SAMESITE = 'Lax' - Adjust according to your application's specific needs while being mindful of security implications.

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 may not be configured with the SameSite attribute, which can expose applications to cross-site request forgery (CSRF) attacks. Ensuring proper cookie settings is essential for protecting user sessions and sensitive data. Remediation Steps Set the SameSite attribute for cookies Ensure that cookies used in your FastAPI application have the SameSite attribute set to either Lax or Strict to mitigate CSRF risks. Example of setting the SameSite attribute ```python from fastapi import FastAPI, Response app = FastAPI() @app.get("/") async def read_root(response: Response): response.set_cookie(key="my_cookie", value="cookie_value", samesite="Lax") return {"message": "Hello World"} ``` Configuration guidance Review your FastAPI application's cookie settings in all routes where cookies are set. Ensure that the samesite parameter is specified in the set_cookie method, and consider using Secure and HttpOnly attributes for added security.

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 SameSite=None attribute for cookies in Django applications can lead to security vulnerabilities such as Cross-Site Request Forgery (CSRF) attacks, as it allows cookies to be sent with cross-origin requests. This can expose sensitive user data and compromise application integrity. Remediation Steps Update Cookie Settings: Ensure that cookies are set with appropriate SameSite attributes to mitigate risks associated with cross-site requests. Example Configuration: In your Django settings, set the cookie attributes as follows: python SESSION_COOKIE_SAMESITE = 'Lax' # or 'Strict' depending on your application needs CSRF_COOKIE_SAMESITE = 'Lax' # or 'Strict' Policy Guidance: Review your application’s cookie usage and ensure that all cookies that do not need to be accessed cross-origin are set to SameSite=Lax or SameSite=Strict. If cookies must be sent across sites, consider implementing additional security measures such as CSRF tokens.

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 missing user in the Dockerfile increases the risk of running containers with root privileges, which can lead to potential escalation of attacks and unauthorized access to sensitive data. Running containers as a non-root user minimizes the impact of a compromised container. Remediation Steps Specify a Non-Root User: Modify the Dockerfile to create and use a non-root user. Example Command: dockerfile RUN useradd -ms /bin/bash myuser USER myuser Policy Guidance: Ensure that all Dockerfiles in your organization include a non-root user setup. Implement a policy that mandates the use of non-root users for all new container images.

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 missing user specification in a Dockerfile can lead to elevated privileges for the container, increasing the risk of unauthorized access and potential exploitation of vulnerabilities. Running containers as the root user can expose the host system to security risks. Remediation Steps Specify a Non-Root User Modify the Dockerfile to create and switch to a non-root user for running applications. Example Command Add the following lines to your Dockerfile: dockerfile RUN useradd -ms /bin/bash myuser USER myuser Policy Guidance Always run containers with a non-root user whenever possible. Update your organization's container security policies to enforce this practice across all Docker images. Consider using tools like Docker Bench Security to audit your Dockerfiles for compliance.