MyESI Automated Compliance & Vulnerability Assessment

Project: myesi-user-service

Generated: 2025-11-20 11:16 UTC **User:** 1

1. Executive Summary

• Compliance Score: **60.0%**

• Average Risk Score: **9.50**

• Total Vulnerabilities: **8**

Standards Compliance:

Standard	Status
ISO_27001	Partially
NIST_SP_800_53	Needs Review
OWASP	Needs Review

2. Compliance Controls Overview

critical — *(General)* — Score: **50.0%**

1 Conduct a comprehensive vulnerability assessment to identify and classify unknown vulnerabilities within your systems.
2 Use tools like Nessus or OpenVAS to scan for vulnerabilities and generate reports on identified risks.
3 Implement a regular patch management policy to ensure all systems and software are updated promptly to address known vulnerabilities. Include guidelines for prioritizing critical and high-severity updates.

moderate — *(General)* — Score: **50.0%**

1 Conduct a Vulnerability Assessment - Identify all systems and applications to determine the specific vulnerabilities present.
2 Apply Security Patches - For systems with known vulnerabilities, ensure that all security patches are applied. For example, on a Linux system, you can update packages using: bash sudo apt update && sudo apt upgrade
3 Review and Update Security Policies - Ensure that your security policies are up to date, including access controls, user permissions, and incident response protocols. Regularly review these policies to align with current best practices.

3. Vulnerability Findings

#	Component	Version	Vuln ID	Severity	Fix / Recommendation
1	**bcrypt**	4.3.0	GHSA-5wg4-74h6-q47v	Moderate	**Update to:** 5.0.0
2	**loguru**	0.7.3	MAL-2025-25559	Critical	### Summary The vulnerability identified as MAL-2025-25559 in the loguru library poses a critical risk, potentially allowing unauthorized access to sensitive logs and information leakage. Immediate action is required to mitigate exploitation. ### Remediation Steps 1. **Update loguru to the latest version**: Ensure you are using a version of loguru that has patched the vulnerability. 2. **Command to update**: ```bash pip install --upgrade loguru ``` 3. **Review and update your logging configuration**: Ensure that sensitive information is not logged and that log files have proper access controls in place. Implement a policy to regularly review and audit logs for sensitive data exposure.
3	**mypy-extensions**	1.1.0	MAL-2024-2685	Critical	### Summary The vulnerability identified as MAL-2024-2685 in the `mypy-extensions` package poses a critical risk, potentially allowing attackers to exploit weaknesses in type checking and lead to unauthorized access or data corruption. ### Remediation Steps 1. **Update the Package**: Immediately update `mypy-extensions` to the latest version that addresses the vulnerability. 2. **Example Command**: Run the following command to update the package via pip: ```bash pip install --upgrade mypy-extensions ``` 3. **Review Dependencies**: Conduct a thorough review of all project dependencies to ensure that no other packages are affected and apply necessary updates or patches as needed. Implement a regular update policy to keep all dependencies current.

4	**pydantic**	2.12.2	MAL-2025-4867	**Critical**	### Summary The vulnerability identified as MAL-2025-4867 in the `pydantic` library poses a critical risk, potentially allowing unauthorized access or manipulation of data. This can lead to data breaches or system compromise if not addressed promptly. ### Remediation Steps 1. **Upgrade the `pydantic` library**: Ensure you are using the latest version of `pydantic` which contains the necessary security patches. 2. **Command to upgrade**: Run the following command in your terminal: ```bash pip install --upgrade pydantic ``` 3. **Review and update your application code**: Check your application for any deprecated features or changes in the API after upgrading. Ensure all usages of `pydantic` are compliant with the latest version guidelines.
5	**pyjwt**	2.10.1	MAL-2025-48036	**Critical**	### Summary The vulnerability identified as MAL-2025-48036 in the pyjwt library poses a critical risk, potentially allowing unauthorized access and exploitation of tokens, leading to data breaches and unauthorized actions. ### Remediation Steps 1. **Upgrade the pyjwt Library** Ensure that you are using the latest version of pyjwt that addresses the vulnerability. 2. **Example Command** To upgrade pyjwt, use the following command: ```bash pip install --upgrade pyjwt ``` 3. **Review Token Handling Policies** Implement strict validation and expiration policies for tokens. Ensure that all tokens are signed with a strong algorithm (e.g., RS256) and check the token's claims properly before processing requests.

6	**python-dotenv**	1.1.1	MAL-2025-48037	**Critical**	### Summary The vulnerability identified as MAL-2025-48037 in the `python-dotenv` library poses a critical risk, potentially allowing unauthorized access to sensitive environment variables, which can lead to data breaches and system compromise. ### Remediation Steps 1. **Update the Library** Upgrade `python-dotenv` to the latest version that addresses the vulnerability. 2. **Example Command** ```bash pip install --upgrade python-dotenv ``` 3. **Configuration Guidance** Review and restrict access to environment files (e.g., `.env`) to ensure that only authorized users and applications can read them. Consider using secure storage solutions for sensitive configurations.
7	**typing-extensions**	4.15.0	MAL-2025-47895	**Critical**	### Summary The vulnerability identified as MAL-2025-47895 in the `typing-extensions` package poses a critical risk, potentially allowing unauthorized access or execution of malicious code. Immediate remediation is essential to protect your systems and data integrity. ### Remediation Steps 1. **Upgrade the `typing-extensions` package** to the latest version where the vulnerability has been patched. 2. **Run the following command** to update the package: ```bash pip install --upgrade typing-extensions ``` 3. **Implement a policy** to regularly check and update dependencies, ensuring that all packages are kept up to date with the latest security patches. Consider using tools like Dependabot or Snyk for automated monitoring.

8	**uvicorn**	0.37.0	MAL-2025-4901	**Critical**	### Summary MAL-2025-4901 identifies a critical vulnerability in the Uvicorn ASGI server, potentially allowing remote code execution or denial of service. This poses a significant risk to applications using Uvicorn, compromising data integrity and availability. ### Remediation Steps 1. **Update Uvicorn**: Immediately upgrade to the latest version of Uvicorn that addresses this vulnerability. 2. **Command to Update**: Run the following command to upgrade Uvicorn: ```bash pip install --upgrade uvicorn ``` 3. **Review Configuration**: Ensure that Uvicorn is configured to run in a secure environment, using the recommended settings for production deployments, such as enabling HTTPS and restricting access to trusted IPs only.

4. Code Findings (Static Analysis)

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

/app/tmp/repos/semgrep-20251120111531/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 pertains to the handling of cookies in FastAPI applications. If cookies are set without the SameSite attribute or with SameSite=None, they may be vulnerable to cross-site request forgery (CSRF) attacks, allowing malicious sites to perform actions on behalf of authenticated users. Remediation Steps Set the SameSite attribute for cookies Ensure that all cookies set in your FastAPI application include the SameSite attribute to mitigate CSRF risks. Example of setting SameSite in FastAPI When creating cookies, specify the samesite parameter: ```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 attribute"} ``` Policy Guidance Adopt a policy to review and enforce cookie settings across your application. All cookies should have the SameSite attribute set to either Lax or Strict unless there is a specific requirement for cross-site access. Regularly audit your application for compliance with this policy.

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

/app/tmp/repos/semgrep-20251120111531/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 on cookies can expose web applications to cross-site request forgery (CSRF) attacks by allowing cookies to be sent with cross-site requests. This can lead to unauthorized actions on behalf of authenticated users. Remediation Steps Review and Update Cookie Settings: Ensure that cookies are configured with the appropriate SameSite attribute. Use SameSite=Lax or SameSite=Strict instead of SameSite=None unless absolutely necessary. Example for Django Cookies: python response.set_cookie('my_cookie', 'value', samesite='Lax') Configuration Guidance: Review your Django settings and ensure that any cookies set for user sessions or sensitive data are configured with the SameSite attribute. If you need cross-site cookie access, ensure that the Secure attribute is also set and consider implementing CSRF protection mechanisms.

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

/app/tmp/repos/semgrep-20251120111531/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 configuration in a Dockerfile can lead to security vulnerabilities, as containers run with root privileges by default. This increases the risk of privilege escalation and potential exploitation if the container is compromised. Remediation Steps Specify a non-root user in your Dockerfile to limit the permissions of the running container. Example command: Add the following lines to your Dockerfile: dockerfile RUN useradd -ms /bin/bash newuser USER newuser Policy Guidance: Ensure that all Dockerfiles in your organization specify a non-root user to run applications. Regularly review Dockerfiles for compliance with this policy.