MyESI Automated Compliance & Vulnerability Assessment

Project: myesi-user-service

Generated: 2025-11-12 10:47 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 Security Assessment Perform a thorough assessment of the system to identify any existing vulnerabilities and misconfigurations.

2 Update Software and Patches Ensure that all software, including operating systems and applications, are up-to-date with the latest security patches. Example command for Linux: bash sudo apt-get update && sudo apt-get upgrade

3 Implement Access Controls Review and enforce access controls to limit user permissions based on the principle of least privilege. Regularly audit user roles and permissions to ensure compliance with security policies. Consider implementing multi-factor authentication (MFA) for all critical systems.

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

1 Conduct a Vulnerability Assessment Identify and assess all systems for vulnerabilities that fall within the moderate category.

2 Patch and Update Software Ensure that all software and systems are updated to the latest versions to mitigate known vulnerabilities. Example command for Linux systems: bash sudo apt-get update && sudo apt-get upgrade

3 Implement Access Controls Review and enforce access controls to limit user permissions based on the principle of least privilege. Ensure that only authorized personnel have access to sensitive data and systems.

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 MAL-2025-25559 in the loguru library can lead to critical security risks, including potential data leakage and unauthorized access to sensitive information. Immediate remediation is essential to protect applications using this library. ### Remediation Steps 1. **Update loguru Library** Upgrade to the latest version of loguru that addresses this vulnerability. Ensure dependencies are also updated to maintain compatibility. 2. **Example Command** ```bash pip install --upgrade loguru ``` 3. **Policy Guidance** Implement a regular review process for third-party libraries and dependencies. Establish a policy for timely updates based on security advisories and vulnerability reports. Consider using automated tools for dependency management to facilitate ongoing compliance.
3	**mypy-extensions**	1.1.0	MAL-2024-2685	**Critical**	### Summary MAL-2024-2685 affects the `mypy-extensions` package, which can lead to potential exploitation due to critical vulnerabilities. This can result in unauthorized access, data breaches, or system compromise if not addressed promptly. ### Remediation Steps 1. **Update the Package** Upgrade `mypy-extensions` to the latest secure version to mitigate the vulnerability. 2. **Example Command** Use the following command to update the package: ```bash pip install --upgrade mypy-extensions ``` 3. **Implement Dependency Monitoring** Regularly monitor and audit dependencies using tools like `pip-audit` or `safety` to identify and remediate vulnerabilities in third-party packages proactively.

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 attackers to exploit input validation flaws, leading to unauthorized data access or manipulation. ### Remediation Steps 1. **Update Pydantic Library** - Ensure you are using the latest version of the Pydantic library, which contains security patches addressing this vulnerability. 2. **Example Command** ```bash pip install --upgrade pydantic ``` 3. **Review and Update Validation Logic** - Review your application’s use of Pydantic for data validation and ensure that all input data is being validated correctly, including edge cases. Implement additional validation rules if necessary to mitigate risks associated with untrusted input.
5	**pyjwt**	2.10.1	MAL-2025-48036	**Critical**	### Summary The vulnerability identified as MAL-2025-48036 affects the `pyjwt` library, which is critical for handling JSON Web Tokens (JWT). Exploitation of this vulnerability could allow unauthorized access to sensitive information and potential compromise of user sessions. ### Remediation Steps 1. **Upgrade the Library**: Immediately update `pyjwt` to the latest stable version that addresses the vulnerability. 2. **Example Command**: Use the following command to upgrade: ```bash pip install --upgrade pyjwt ``` 3. **Review Configuration**: Ensure that JWT secrets are stored securely and are of sufficient length and complexity. Implement proper token expiration and validation checks in your application to mitigate the impact of potential token misuse.

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 could lead to data breaches and system compromise. ### Remediation Steps 1. **Upgrade the Library** Update the `python-dotenv` library to the latest version that addresses this vulnerability. 2. **Example Command** ```bash pip install --upgrade python-dotenv ``` 3. **Configure Environment Variables Securely** Ensure that sensitive environment variables are not stored in plain text within your application. Use secure vaults or encrypted storage solutions to manage 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 due to potential exploitation that could lead to unauthorized access or execution of arbitrary code. Immediate action is required to mitigate this risk. ### Remediation Steps 1. **Update the `typing-extensions` Package** - Ensure you are using the latest version of the `typing-extensions` package, which contains patches for the identified vulnerabilities. 2. **Example Command** ```bash pip install --upgrade typing-extensions ``` 3. **Configuration Guidance** - Review your dependency management practices to ensure that all packages, including `typing-extensions`, are regularly updated. Implement automated dependency checks using tools like `pip-audit` or `safety` to monitor for known vulnerabilities.

8	**uvicorn**	0.37.0	MAL-2025-4901	**Critical**	### Summary The vulnerability MAL-2025-4901 in the Uvicorn server poses a critical risk, potentially allowing remote code execution or denial of service. This can lead to unauthorized access to sensitive data or complete service disruption. ### Remediation Steps 1. **Update Uvicorn**: Ensure that you are running the latest version of Uvicorn, as vulnerabilities are often patched in new releases. 2. **Command to Update**: ```bash pip install --upgrade uvicorn ``` 3. **Review Configuration**: Ensure Uvicorn is configured to run behind a reverse proxy (e.g., Nginx or Apache) and restricts access to trusted IPs only. Additionally, consider enabling security features such as CORS and CSRF protection.

4. Code Findings (Static Analysis)

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

/app/tmp/repos/semgrep-20251112094210/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 in your FastAPI application are not set with the SameSite attribute, which may expose your application to cross-site request forgery (CSRF) attacks. While the severity is informational, it is crucial to implement proper cookie configurations to enhance security. Remediation Steps Set the SameSite attribute for cookies: Ensure that all cookies are configured with the SameSite attribute to prevent CSRF attacks. Example of setting SameSite in FastAPI: ```python from fastapi import FastAPI, Response app = FastAPI() @app.get("/") async def set_cookie(response: Response): response.set_cookie(key="my_cookie", value="cookie_value", samesite="Lax") return {"message": "Cookie set with SameSite attribute"} `` 3. **Configuration guidance**: Review all cookie settings in your FastAPI application and adjust theSameSiteattribute to eitherLaxorStrictbased on your application's requirements. Avoid usingNone` unless absolutely necessary and ensure you are using HTTPS if you do.

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

/app/tmp/repos/semgrep-20251112094210/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 are being set without the SameSite attribute, which can lead to cross-site request forgery (CSRF) attacks. This is particularly concerning when cookies are set with Secure and SameSite=None, as it allows cross-origin requests in an insecure manner. Remediation Steps Update Cookie Settings: Ensure that cookies are configured with the appropriate SameSite attribute to enhance security. Example Configuration: python # In your Django settings.py SESSION_COOKIE_SAMESITE = 'Lax' # or 'Strict' based on your application's needs CSRF_COOKIE_SAMESITE = 'Lax' # or 'Strict' Policy Guidance: Review and adjust your cookie policy to ensure that sensitive cookies do not have SameSite=None unless absolutely necessary. Always pair with the Secure attribute if using SameSite=None.

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

/app/tmp/repos/semgrep-20251112094210/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 indicates that the Dockerfile does not specify a non-root user for running containers, which can lead to privilege escalation and increased attack surface if the container is compromised. Remediation Steps Add a Non-Root User: Modify the Dockerfile to create and use a non-root user for running the application. Example Command: dockerfile RUN useradd -ms /bin/bash appuser USER appuser Configuration Guidance: Always ensure that the Dockerfile specifies a non-root user with the USER directive after the application and dependencies are installed. This practice minimizes security risks associated with running containers as root.