MyESI Automated Compliance & Vulnerability Assessment

Project: myesi-user-service

Generated: 2025-11-12 10:40 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. Identify the Vulnerability Conduct a thorough assessment to pinpoint the source of the vulnerability using vulnerability scanning tools.
2. Apply Security Patches Update all affected systems with the latest security patches and firmware updates. Example command for updating packages on a Debian-based system: bash sudo apt-get update && sudo apt-get upgrade
3. Review and Harden Configuration Ensure that system configurations adhere to security best practices. Disable unnecessary services and ports. For example, if using a firewall, ensure that only required ports are open: bash sudo ufw allow [port_number] # Allow specific port sudo ufw deny [port_number] # Deny access to all other ports

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

1. Perform a vulnerability assessment. - Regularly scan your systems to identify potential vulnerabilities using tools like Nessus or OpenVAS.
2. Apply relevant patches and updates. - Ensure all software, including operating systems and applications, are updated to the latest versions. Use the command: bash sudo apt update && sudo apt upgrade
3. Review and strengthen access controls. - Implement the principle of least privilege (PoLP). Ensure users have only the permissions necessary for their roles. Regularly review and adjust user roles and permissions in your access management system.

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 This control requires attention in General. ### Remediation Steps - Immediately patch or isolate affected components. Deploy emergency fixes and block exploit paths. - Validate compliance and re-run assessment.
3	**mypy-extensions**	1.1.0	MAL-2024-2685	**Critical**	### Summary The vulnerability MAL-2024-2685 in the \`mypy-extensions\` library poses a significant risk, potentially allowing attackers to execute arbitrary code or manipulate data. Immediate action is required to mitigate the threat. ### Remediation Steps 1. **Update the \`mypy-extensions\` library to the latest version** Ensure you are using the most recent version that addresses the vulnerability. 2. **Example command to update** \`\`\`bash pip install --upgrade mypy-extensions \`\`\` 3. **Review and test your application's dependencies** Conduct a thorough review of all dependencies to ensure compatibility with the updated version and run regression tests to confirm that the update does not introduce new issues.

4	**pydantic**	2.12.2	MAL-2025-4867	**Critical**	### Summary The vulnerability identified as MAL-2025-4867 in the Pydantic library allows for potential data validation bypass, leading to critical security risks including data corruption and unauthorized access to sensitive information. ### Remediation Steps 1. **Upgrade Pydantic**: Update the Pydantic library to the latest stable version where the vulnerability has been patched. - **Example Command**: ```bash pip install --upgrade pydantic ``` 2. **Review and Refactor Code**: Audit your codebase to ensure that any instances of Pydantic usage are properly validating data and handling exceptions. 3. **Implement Additional Validation**: Consider implementing additional data validation layers or using stricter validation rules in your Pydantic models to mitigate potential risks. - **Configuration Guidance**: Review your Pydantic model definitions and ensure you are using `@validator` and `@root_validator` effectively to enforce data integrity and security.
5	**pyjwt**	2.10.1	MAL-2025-48036	**Critical**	```markdown ### Summary The identified vulnerability in the `pyjwt` library (MAL-2025-48036) poses a critical risk, potentially allowing attackers to exploit weaknesses in JSON Web Token (JWT) handling, leading to unauthorized access or data breaches. ### Remediation Steps 1. **Upgrade the Library** Update `pyjwt` to the latest version where the vulnerability is patched. 2. **Example Command** Run the following command to upgrade the library: ```bash pip install --upgrade pyjwt ``` 3. **Configuration Guidance** Ensure that your JWT implementation follows best practices: - Use strong secret keys for signing tokens. - Implement token expiration and validation checks. - Consider using `algorithms=["HS256"]` or stronger for token signing. ```

6	**python-dotenv**	1.1.1	MAL-2025-48037	**Critical**	### Summary The vulnerability identified as MAL-2025-48037 in the `python-dotenv` library can lead to the exposure of sensitive environment variables, potentially allowing unauthorized access to sensitive data and systems. ### Remediation Steps 1. **Upgrade the Library**: Immediately update `python-dotenv` to the latest version to mitigate the vulnerability. 2. **Command to Upgrade**: Run the following command to upgrade: ```bash pip install --upgrade python-dotenv ``` 3. **Configuration Guidance**: Review and restrict access to environment variable files (e.g., `.env`). Ensure that these files are not included in version control systems (e.g., by adding them to `.gitignore`) and that they have proper permissions set to limit access.
7	**typing-extensions**	4.15.0	MAL-2025-47895	**Critical**	### Summary MAL-2025-47895 affects the `typing-extensions` package, which is critical due to potential vulnerabilities that could lead to arbitrary code execution or data leakage in applications that depend on it. Immediate action is required to mitigate risks. ### Remediation Steps 1. **Upgrade the `typing-extensions` package to the latest version.** 2. **Command to upgrade:** ```bash pip install --upgrade typing-extensions ``` 3. **Verify compatibility:** Ensure that the new version of `typing-extensions` is compatible with your application and other dependencies. Test your application in a staging environment before deploying to production.
8	**uvicorn**	0.37.0	MAL-2025-4901	**Critical**	### Summary This control requires attention in General. ### Remediation Steps - Immediately patch or isolate affected components. Deploy emergency fixes and block exploit paths. - Validate compliance and re-run assessment.

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 This control requires attention in security. Remediation Steps Investigate severity and verify exploitability before mitigation planning. Validate compliance and re-run assessment.

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:

```markdown Summary The SameSite=None attribute for cookies in Django can lead to security vulnerabilities, particularly around Cross-Site Request Forgery (CSRF) attacks. This setting allows cookies to be sent in cross-origin requests, which may expose sensitive user data. Remediation Steps Review Cookie Settings Assess your existing cookie settings and determine if the SameSite=None attribute is necessary for your application. Update Cookie Attributes If SameSite=None is not required, change the cookie settings to use either SameSite=Lax or SameSite=Strict. Example command in Django settings: python SESSION_COOKIE_SAMESITE = 'Lax' # or 'Strict' Set Secure Flag Ensure that cookies with SameSite=None also have the Secure flag set, which restricts cookies to be sent over HTTPS only. Configuration guidance: python SESSION_COOKIE_SECURE = True Test Changes After making the changes, test your application thoroughly to ensure that functionality is not adversely affected and that cookies are being set with the correct attributes. ```

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 absence of a non-root user in the Dockerfile can lead to security vulnerabilities, as running containers as the root user can provide attackers with elevated privileges if a container is compromised. Remediation Steps Create a Non-Root User Add a non-root user to the Dockerfile to minimize potential security risks. Example Command Use the following commands in your Dockerfile: Dockerfile RUN addgroup --system myusergroup && adduser --system --ingroup myusergroup myuser USER myuser Configuration Guidance Ensure that the application and any required files have the appropriate permissions set for the non-root user to operate correctly. This can be done by setting the ownership of files and directories within the Dockerfile: Dockerfile RUN chown -R myuser:myusergroup /path/to/application Always test the container to verify that it functions properly under the new user context.