Project Presentation

BY: AISHAT SUBAIR

Secure Coding Practices for Flask Applications

Addressing Flask Debug Mode Vulnerability

Problem Overview

•Issue: Flask app running with debug=True
•Severity: High
•Confidence: Medium
•Risk: Exposes Werkzeug debugger, allowing arbitrary code execution.
•CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)

Why It's a Problem

•1. Werkzeug Debugger provides detailed error pages with an interactive shell.

•2. Exposes the application to arbitrary code execution.

3. Attackers can exploit this to compromise the server and data

Recommendations

•1. Never use debug=True in production.

•2. Use environment variables to control debug mode.

•3. Deploy with a production-grade WSGI server (e.g., Gunicorn).

•4. Restrict access to the debugger in development.

5. Use a configuration management system

Disabling Debug Mode

- Ensure debug mode is disabled in production:
-
- if __name__ == '__main__':
-   app.run(debug=False)

Environment-Based Configuration

- Use environment variables for flexible configuration:
  -
  - import os
  - if __name__ == '__main__':
  - debug_mode = os.getenv('FLASK_DEBUG', 'False').lower() == 'true'
  - app.run(debug=debug_mode)

Deploying with Production Servers

•Avoid using Flask's built-in server in production:

•

•Deploy with Gunicorn:

```
•gunicorn -w 4 -b 0.0.0.0:8000 app:app
```

Configuration Management

- Separate configurations for different environments:
-
- # config.py
- class Config:
- DEBUG = False
- class DevelopmentConfig(Config):
- DEBUG = True
- class ProductionConfig(Config):
- DEBUG = False

Summary

•Disable debug=True in production.

•. Use environment variables for dynamic debug settings.

•3. Deploy with production-grade servers.

•4. Restrict debugger access in development.

•5. Regularly review deployment scripts for security.

Files

FILES FOR THE TEST PRACTICES

Thank You