Student Project Lab: Application Security Requirements

Purpose

All applications published through Student Project Lab are publicly accessible software running on shared department-managed infrastructure.

To protect users, project developers, other hosted applications, and the hosting environment, every application must satisfy the security requirements described here.

All applications must pass security review before publication.

The requirements apply to initial releases and subsequent updates. Approval of one release does not automatically authorize future releases.

These requirements are informed by the OWASP Application Security Verification Standard (ASVS), with a baseline adapted to the risks and resources of student-developed applications. Passing the Student Project Lab review does not constitute OWASP certification or guarantee that an application is free of vulnerabilities.

1. Secure Development

Applications must follow recognized secure software development practices.

Requirements

- Use maintained frameworks, libraries, and dependencies.
- Correct known vulnerabilities that present unacceptable risk before publication.

- Use established, maintained security libraries instead of implementing custom cryptographic algorithms or authentication mechanisms without approval.
- Validate untrusted inputs on the server.
- Use parameterized database queries or a secure ORM to prevent SQL injection.
- Apply context-appropriate output encoding to prevent cross-site scripting.
- Protect state-changing requests against cross-site request forgery where applicable.
- Avoid insecure deserialization and unsafe execution of user-controlled input.
- Handle errors without exposing internal implementation details.

Students must disclose unresolved security findings and known vulnerabilities during review.

2. Authentication and Authorization

Applications with user accounts must implement appropriate authentication and authorization controls.

Requirements

- Passwords must never be stored in plaintext or using fast, general-purpose hashes.
- Password authentication must use an established password-hashing mechanism supported by the framework, such as Argon2id or an appropriately configured alternative.
- Authentication credentials must be transmitted securely.
- Sessions must use appropriately configured secure cookies or another reviewed mechanism.
- Authentication endpoints must have appropriate protections against brute-force attacks.
- Sensitive operations must require authentication.
- Authorization must be enforced on the server for every protected operation.
- Users must not be able to access another user's private information merely by changing identifiers or URLs.
- Administrative functionality must be restricted to authorized administrators.
- Default passwords and demonstration credentials must not remain active in production.

Students must test access control using accounts with different permission levels.

Applications using third-party identity providers must document the integration and its security assumptions.

3. Secrets and Sensitive Information

Secrets must never be embedded in source code, container images, or repository history.

This includes:

- Passwords.
- Database connection credentials.
- API keys.
- Authentication tokens.
- Application encryption and signing keys.
- Private cryptographic keys.

Use the deployment platform's approved secret configuration.

Applications must not log passwords, session tokens, authorization headers, or other sensitive credentials.

Publicly hosted applications must not collect or store actual student records, protected educational information, medical information, financial account information, or other sensitive personal data without explicit prior authorization and an approved protection plan.

Synthetic or fictional demonstration data is preferred.

Do not populate demonstration databases with real information copied from university systems, workplaces, or other individuals.

4. Production Configuration

Applications must be configured for production operation.

Requirements

- Debug mode must be disabled.
- Detailed stack traces must not be exposed to visitors.
- Development-only endpoints and debugging tools must be disabled.
- Unnecessary services and ports must not be publicly accessible.
- HTTPS must be used for public access.
- Session cookies must use appropriate security attributes.

- Cross-origin requests must be restricted to approved origins.
- Security-related HTTP headers must be configured appropriately.
- Upload limits, request limits, and resource constraints must be reasonable for the application.
- Configuration failures must not silently enable insecure defaults.

The hosting platform provides HTTPS termination. Applications must still correctly handle forwarded protocol information and securely configure authentication and sessions.

5. Container Security

Student applications share infrastructure with other hosted projects. Container security is therefore mandatory.

Prohibited configurations

The following are prohibited unless the department explicitly authorizes an exception:

- Privileged containers.
- Access to the Docker socket.
- Host filesystem mounts outside explicitly approved storage locations.
- Host networking.
- Access to host devices.
- Excessive Linux capabilities.
- Running unnecessary services as root.
- Publicly exposed database or cache ports.
- Container configurations intended to bypass platform restrictions.

Applications should use non-root container users wherever feasible.

Images must come from trusted sources and must not contain unnecessary software or embedded credentials.

The department may impose CPU, memory, disk, network, or process limits.

6. Files and Persistent Storage

Applications that accept file uploads must:

- Restrict allowed file types and sizes.
- Validate file content where appropriate, rather than trusting file extensions alone.

- Store uploaded files safely.
- Prevent directory traversal.
- Prevent uploaded files from executing as application code.
- Enforce authorization when accessing private files.
- Avoid exposing secrets or internal server files through download functionality.

Applications must not rely on the container's temporary filesystem for persistent application data.

Persistent storage must be declared and documented.

7. Database Security

Applications using databases must:

- Use separate, appropriately limited database credentials.
- Avoid using database administrator accounts for normal application requests.
- Avoid publishing database ports to the internet.
- Use parameterized queries or safe ORM operations.
- Protect sensitive records through appropriate authentication and authorization.
- Handle migrations and schema changes predictably.
- Document database initialization and restoration procedures.

Development and production databases must remain separate.

8. Third-Party Services and Network Access

All external dependencies must be documented.

Examples include payment services, mapping APIs, email delivery, external databases, cloud storage, and AI APIs.

Applications may only use external services for which the student or department has appropriate authorization.

The following activities require specific approval and may be prohibited on the shared platform:

- Network scanning or penetration-testing services.
- Automated vulnerability exploitation.
- Proxies or relays accessible to arbitrary internet users.

- Automated scraping or high-volume external requests.
- Cryptocurrency mining.
- Hosting malware or other potentially dangerous executable content.
- Unrestricted remote command execution.
- Hosting infrastructure that requires privileged operating-system access.

Some cybersecurity research projects may be legitimate academic work but unsuitable for shared public hosting. The department may require a separate isolated environment instead.

9. Security Testing

Students must perform reasonable security testing before requesting review.

At minimum:

- Run the application's automated tests.
- Perform dependency vulnerability scanning.
- Check for committed secrets.
- Review container and deployment configuration.
- Test authentication and authorization, where applicable.
- Test the application's input validation.
- Review error handling and logging.
- Confirm that debugging features are disabled.
- Verify that only intended HTTP services are externally accessible.

Recommended tools include:

- GitHub Dependabot or an equivalent dependency scanner.
- GitHub CodeQL or another suitable static analyzer.
- Gitleaks or another secret scanner.
- Trivy or another container vulnerability scanner.
- OWASP ZAP for application security testing.

Students may use alternative tools if they provide comparable coverage.

Testing must be limited to applications and systems the student owns or is explicitly authorized to assess.

10. Requirements for Higher-Risk Features

Some application features require additional review because they introduce greater security or privacy risk.

Students must identify these features in their publication request.

Examples include:

- Public account registration.
- User-uploaded files.
- User-generated HTML or executable content.
- Administrative dashboards.
- Payment processing or financial transactions.
- Third-party OAuth or SSO.
- Email sending or other externally triggered communication.
- Sensitive personal information.
- Public APIs allowing data modification.
- AI agents capable of using external tools or executing code.
- Integration with physical devices or external systems.
- Security testing, network analysis, or similar cybersecurity functionality.

The department may require additional testing, architectural changes, isolation, or external services before publication.

11. Vulnerability Severity and Approval

Security findings will be evaluated based on exploitability, impact, exposure, and the application's purpose.

The department may use CVSS alongside contextual risk analysis when classifying vulnerabilities.

Severity	Publication requirement
Critical	Must be resolved before publication
High	Must be resolved before publication
Medium	Must be resolved or receive documented risk acceptance
Low	May be accepted with documented remediation recommendations
Informational	Document where appropriate

Severity classifications are determined by the reviewer, not solely by automated scanning tools.

A finding marked as a false positive must be explained and accepted by the reviewer.

A successful automated security scan does not replace manual review.

12. Review Outcomes

Security reviews may result in:

Approved: The identified commit meets the publication requirements.

Approved with conditions: Publication is permitted only after specified conditions are satisfied and verified.

Changes required: The student must correct identified deficiencies and submit a new release candidate.

Rejected: The application is unsuitable for the shared hosting environment or presents unacceptable risk.

The department reserves the right to reject, suspend, or remove applications that violate these requirements.

13. Changes After Approval

Security approval is associated with an exact source-code revision and deployment configuration.

Any code change after approval may require a new review.

Changes involving authentication, permissions, dependencies, API behavior, database access, infrastructure configuration, or sensitive data handling will generally require renewed security assessment.

Students may not bypass this process by pushing new tags, altering deployment files, or attempting to access production directly.

14. Security Review Checklist

Students must complete the following checklist before requesting publication.

Source code

- Dependencies are reasonably current and supported.
- No known critical or high-risk vulnerabilities remain unresolved.
- Untrusted input is validated.
- Database queries are protected against injection.
- Application output is appropriately encoded.
- Error handling does not reveal sensitive information.

Authentication and authorization

- Passwords are securely hashed, if applicable.
- Session management is secure.
- Authorization is enforced server-side.
- Administrative functionality is protected.
- Default credentials are removed or disabled.

Secrets and privacy

- No credentials are committed to the repository.
- Production secrets are externally configured.
- Logs do not expose sensitive credentials.
- Demonstration data is synthetic or otherwise authorized.
- Any personal data collection has been disclosed.

Deployment

- Containers do not require privileged access.
- The application uses production settings.
- Debug features are disabled.
- Databases and internal services are not publicly exposed.
- Persistent storage is configured appropriately.
- Health checks and startup procedures work.
- Required external network access is documented.

Verification

- Automated tests pass.
- Dependency and secret scans have been performed.
- Security-relevant functionality has been tested.

- Known risks and limitations are disclosed.
- The exact release candidate has been prepared.

Security review is a publication requirement, not a guarantee of security. Students remain responsible for promptly reporting newly discovered vulnerabilities in their applications.

Revision #1
Created 2026-11-06 18:03:37 UTC by jnewton1024
Updated 2026-11-06 18:07:01 UTC by jnewton1024