

# Application Security Coding Guidelines

**Ministry of SaskBuilds and Procurement**  
Information Technology Division  
Cybersecurity and Risk Management Branch

March 6, 2026

# Application Security Coding Guidelines

Ministry of SaskBuilds and Procurement  
Information Technology Division, Cybersecurity and Risk Management Branch

## Confidentiality Statement

This document is intended for the Saskatchewan Ministry of SaskBuilds and Procurement, Information Technology Division, and its partner organizations. It may contain legally privileged and/or confidential information and must not be disseminated, distributed, or copied.

## Contents

REFERENCES .....	1
1. OVERVIEW .....	2
2. OWASP TOP 10:2025.....	2
3. SECURE CODING PRINCIPLES.....	3
4. SECURE CODING PRACTICES.....	4
REVISION HISTORY .....	4

## References

The OWASP Top 10 information contained in this document is found with additional information and detail on the OWASP.org website and is © Copyright 2021-2025 - OWASP Top 10 Team.

[OWASP Top 10:2025](#) (link is external).

# Application Security Coding Guidelines

Ministry of SaskBuilds and Procurement  
Information Technology Division, Cybersecurity and Risk Management Branch

## 1. Overview

It is necessary for Application Developers to be able to identify and understand types of vulnerabilities in existence that place applications at high risk due to programming defects. Special consideration given to these areas will result in the highest probability of reducing the threat to an application of being exploited by a programming defect. Though there are many defects in existence that have security implications, it is generally agreed to that the OWASP Top 10 comprise most of the security breaches that occur.

The OWASP Top 10 is standard awareness for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

## 2. OWASP Top 10:2025

The OWASP Top 10:2025 identifies the most critical web application security risks and reflects updated data and industry trends. All links in the following table redirect to the OWASP.org website.

OWASP Top 10:2025	Description
<a href="#"><u>A01:2025 – Broken Access Control</u></a>	Failures that allow attackers to access data or functionality without proper authorization.
<a href="#"><u>A02:2025 – Security Misconfiguration</u></a>	Misconfigured permissions, default settings, unnecessary features, or inconsistent configurations.
<a href="#"><u>A03:2025 – Software Supply Chain Failures</u></a>	New category expanding on vulnerable/outdated components; covers dependency, CI/CD, and build-system risks.
<a href="#"><u>A04:2025 – Cryptographic Failures</u></a>	Weak or incorrect use of cryptography, outdated algorithms, or improper key management.
<a href="#"><u>A05:2025 – Injection</u></a>	Classic injection flaws (SQLi, command injection, etc.) due to unsanitized or untrusted input.
<a href="#"><u>A06:2025 – Insecure Design</u></a>	Flaws stemming from missing or weak security controls at the design/architecture level.
<a href="#"><u>A07:2025 – Authentication Failures</u></a>	Issues with authentication processes (broken logins, session problems, credential weaknesses).
<a href="#"><u>A08:2025 – Software or Data Integrity Failures</u></a>	Risks involving untrusted deserialization, unsigned code, or tampered data.
<a href="#"><u>A09:2025 – Security Logging and Alerting Failures</u></a>	Poor or missing logging, alerting, and monitoring that hinder threat detection and response.
<a href="#"><u>A10:2025 – Mishandling of Exceptional Conditions</u></a>	New category covering errors, fail-open scenarios, and improper exception handling.

# Application Security Coding Guidelines

Ministry of SaskBuilds and Procurement  
Information Technology Division, Cybersecurity and Risk Management Branch

## 3. Secure Coding Principles

The following secure coding principles should be observed:

- **Principle of Least Privilege** – Do not require the application to run on the administrator account. Use coding discipline to determine what privileges are necessary and explicitly grant only those privileges to the non-administrator account upon which the application runs.
- **Principle of Least Trust** – Do not trust input provided by external users. Assume that external systems are insecure.
- **Principle of Simplicity** – Ensure that security subsystems are manageable and not overly complicated for users and administrators. Large interfaces and complex solutions run a higher risk for the existence of security vulnerabilities than small interfaces and simple code. The more entrance points made available to an application and the more intricate the application’s functionality, the higher the potential for defects. Some of those defects will be security related. With respect to security functionality added to an application, it must be easy to install, configure, and use, or it will be disabled or bypassed.
- **Avoid Security Through Obscurity** – Assume source code will be inspected by hackers. Design applications with this in mind. “Secrets” such as hidden fields, path names, etc. may slow down an attacker but they won’t stay secret forever. Application security based on “security by obscurity” is destined for failure.
- **Avoid a Single Point of Failure** – An application should not be designed in such a fashion that if a single mechanism such as a firewall or an authentication service fails, the entire application is placed at risk. Another name for this principle is Defense in Depth. If one mechanism fails, there should be a second mechanism that must be defeated before the application can be compromised. If the second mechanism fails, there is a third, etc. A DMZ is an example of not having a single point of failure. If the outer firewall fails, though the web server may be vulnerable and compromised, the rest of the application and data is behind a second firewall and is, therefore, still protected.
- **Data must be vetted** – Inspect every return code from every function call. This includes system library routines.
- **Separation of Services** – Dedicate a single service to a single platform. Though it is tempting from a cost perspective to run multiple services on the same platform doing such increases the overall complexity of the system and therefore increases the risk of security vulnerabilities.
- **Secure Defaults** – Do not enable services by default unless it is necessary. By default, things should be disabled unless they are explicitly enabled by a decision. The default settings should be the secure mode of operation. Security is not something that should have to be “turned on,” it should be always present unless explicitly disabled.
- **Fail to a secure mode** – Ensure that applications, systems, and subsystems fail in a secure manner. This is called failing closed as opposed to failing open. Code must be written to verify explicitly what is allowed before allowing it. Do not attempt to check for the cases where things are disallowed, an event may be missed the application could fail in a non-secure mode because the failure was not recognized.

# Application Security Coding Guidelines

Ministry of SaskBuilds and Procurement  
Information Technology Division, Cybersecurity and Risk Management Branch

## 4. Secure Coding Practices

Secure coding practices must be incorporated into all life cycle stages of an application development process. The following minimum set of secure coding practices should be implemented when developing and deploying covered applications:

1. Formalize and document the software development life cycle (SDLC) processes to incorporate a major component of development process:
  - [Requirements](#) (link is external)
  - [Architecture and Design](#) (link is external)
  - [System Integration](#) (link is external)
  - [Testing](#) (link is external)
  - [Deployment](#) (link is external)
  - [Maintenance](#) (link is external)

While there is no GoS standard or prescriptive model for SDLC methodologies, the project development team should ensure the above major components of a development process are defined in respect to the adopted development methodology, which could be traditional waterfall model, agile or other models.

2. Integrate [secure coding principles](#) into SDLC components by providing a general description of how the secure coding principles are addressed in Architecture and Design documents. If a secure coding principle is not applicable to the project, this should be explicitly documented along with a brief explanation.
3. Custom code should be reviewed using automated or manual analysis.
4. Code review should follow [OWASP Code Review Guide](#) (link is external).
5. Perform automated application security testing as part of the overall application testing process.
6. Development and testing environments should redact all sensitive data or use de-identified data.

## Revision History

Version ID	Date of Change	Author	Rationale
0.1	18 April 2016	Fuad Iddrisu	Draft created.
1.0	02 May 2016	Fuad Iddrisu	Version 1.0 Finalized.
1.1	22 November 2018	Kelsey Sproat	Transferred document into new visual format. Updated OWASP Top 10. Sent for peer review.
2.0	02 January 2019	Kelsey Sproat	Peer review complete. All revisions accepted. Version 2.0 Finalized.
3.0	08 March 2022	Kulbir Jaglan	Updated content as per OWASP 2021 guidelines. Version 3.0 Finalized.
3.1	06 March 2026	Kelsey Sproat	Minor formatting revisions. Updated content as per OWASP 2025 guidelines. Verified and updated broken links to external sources.