Application Security Coding Guidelines

Ministry of SaskBuilds and Procurement
Information Technology Division
Information Security Branch

Last revised: March 2022
Last reviewed: March 2022
Next review: March 2023

Government of Saskatchewan

Table of Contents

Revision History 2
1. Introduction 3
2. OWASP Top 10 Vulnerabilities 3
3. Secure Coding Principles 4
4. Secure Coding Practices 5
5. References 6

Revision History

Date (dd/mm/yyyy)	**Version**	**Comments**	**Reviewers Name**
18/04/2016	0.1	Draft created	Fuad Iddrisu
02/05/2016	1.0	Version 1 finalized.	Fuad Iddrisu
22/11/2018	1.1	Transferred document into new visual format. Updated OWASP Top 10. Sent for peer review.	Darren Sproat
02/01/2019	2.0	Peer review complete. No revisions.	Darren Sproat
08/03/2022	3.0	Updated Content as per OWASP 2021 guidelines	Kulbir Jaglan

1. Introduction

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.

2. OWASP Top 10 Vulnerabilities

OWASP Top10 2021	Description
A1: 2021 Broken Access Control	Injection flaws such as SQL, NoSQL, OS, and LDAP injection occur when untrusted data is sent to an interpreted as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2: 2021 Cryptographic Failures	Many web applications and APIs do not properly protect sensitive data such as financial, health, and personally identifiable information (PII). Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection such as encryption at-rest or in-transit and requires special precautions when exchanged with the browser.
A3: 2021 Injection	Injection flaws such as SQL, NoSQL, OS, and LDAP injection occur when untrusted data is sent to an interpreted as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A4 : 2021 Insecure Design	Insecure design is a wide term that encompasses a variety of flaws and is defined as "missing or poor control design." Threat modeling, secure design patterns, and reference architectures are among the new categories for 2021, with a demand for increasing the usage of threat modeling, safe design patterns, and reference architectures
A5: 2021 Security Misconfiguration	Security misconfiguration is the most seen issues. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A6: 2021 Vulnerable and Outdated Components	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A7: 2021 Identification and Authentication Failures	When applications incorrectly execute functions related to session management or user authentication, intruders may be able to compromise passwords, security keys, or session tokens and permanently or temporarily assume the identities and permissions of other users. This vulnerability poses a grave threat to the security of the application and the resources it accesses and can also severely compromise other assets connected to the same network.

OWASP Top10 2021	Description
A8: 2021 Software and Data Integrity Failures	Code and infrastructure that do not guard against integrity violations are referred to as software and data integrity failures. A program that uses plugins, libraries, or modules from untrusted sources, repositories, or content delivery networks (CDNs) is an example of this. Unauthorized access, malicious code, or system compromise can all be risks of an unsecured CI/CD pipeline. Finally, many programs now have auto-update capabilities that allow updates to be obtained without necessary integrity checks and applied to previously trusted applications. Attackers could potentially distribute and run their own updates across all systems with this functionality.
A9: 2021 Security Logging and Monitoring Failures	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.
A10: 2021 Server-Side Request Forgery (SSRF)	SSRF is a web security flaw that allows an attacker to force a server-side application to send HTTP requests to any domain the attacker chooses. When a web application fetches a remote resource without validating the user-supplied URL, an SSRF fault occurs. Even if the program is secured by a firewall, VPN, or another sort of network access control list, an attacker can force it to send a forged request to an unexpected location

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.

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)
   - Implementation (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.

5. References

1. 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 – OWASP Top 10 team.

   OWASP Top 10 – 2021, The Ten Most Critical Web Application Security Risks (link is external).

2. OWASP Secure Coding Guidelines (link is external).
3. OWASP Code Review Guide (link is external).