Application Security Coding Guidelines

Ministry of Central Services
Information Technology Division
Information Security Branch

Last revised: November 2018
Last reviewed: November 2018
Next review: November 2019

Government of Saskatchewan

Table of Contents

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

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 the majority of the security breaches that occur.

2. Top 10 OWASP Vulnerabilities

OWASP Top10 2017	Description
A1: 2017 **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.
A2: 2017 **Broken Authentication**	Application functions related to authentication and session management are often implemented incorrectly allowing attackers to compromise passwords, keys, or session tokens or to exploit other implementation flaws to assume other users' identities temporarily or permanently .
A3: 2017 **Sensitive Data Exposure**	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.
A4: 2017 **XML External Entities (XXE)**	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5: 2017 **Broken Access Control**	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6: 2017 **Security Misconfiguration**	Security misconfiguration is the most commonly seen issue. 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.
A7: 2017 **Cross-site Scripting (XSS)**	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8: 2017 **Insecure Deserialization**	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

OWASP Top10 2017	Description
A9: 2017 Using Components with Known Vulnerabilities	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.
A10: 2017 Insufficient Logging and Monitoring	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.

3. Application Security Coding Guidelines

The following application coding guidelines 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 actually 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 absolutely 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. 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 © 2003 – 2017 The OWASP Foundation.

OWASP Top 10 – 2017, The Ten Most Critical Web Application Security Risks