APPLICATION SECURITY CODING GUIDELINES

1.0 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.

OWASP TOP 10 web application security vulnerabilities

A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter 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-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. 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.
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to

	date.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
A10-Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

TABLE SOURCE: 2013 OWASP Foundation

2.0 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.

Revision History

Review Date	Comments	Reviewers Name
April 18, 2016	Draft guidelines	Fuad Iddrisu
May 02, 2016	Version 1	Fuad Iddrisu