Australian Government Information Security Manual

Guidelines for software development

Application development

Development environments

Segregating software development, testing and production environments can limit the spread of malicious code and minimises the likelihood of faulty code in a production environment.

Security Control: 0400; Revision: 4; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Software development, testing and production environments are segregated.

Security Control: 1419; Revision: 1; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Development and modification of software only takes place in development environments.

Security Control: 1420; Revision: 2; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Must
Information in production environments is not used in testing or development environments unless the testing or development environments are secured to the same level as the production environments.

Security Control: 1422; Revision: 3; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Unauthorised access to the authoritative source for software is prevented.

Secure software design

Threat modelling is an important part of secure software design. Threat modelling identifies at risk components of software, enabling security controls to be identified to reduce security risks.

Security Control: 1238; Revision: 3; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Threat modelling and other secure design techniques are used to ensure that threats to software and mitigations to those threats are identified and accounted for.

Secure programming practices

Once a secure software design has been identified, secure programming practices should be followed during software development activities.

Security Control: 0401; Revision: 3; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Platform-specific secure programming practices are used when developing software, including using the lowest privilege needed to achieve a task, checking return values of all system calls and validating all inputs.

Software testing

Software testing will lessen the possibility of security vulnerabilities in software being introduced into a production environment. Software testing can be performed using both static testing, such as code analysis, as well as dynamic testing, such as input validation and fuzzing. Using an independent party for software testing will remove any bias that can occur when a software developer tests their own software.

Security Control: 0402; Revision: 3; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Software is tested for security vulnerabilities by software developers, as well as an independent party, before it is used in a production environment.

Further information

An example of a secure development life cycle model, known as the Trustworthy Computing Security Development Lifecycle, and used by Microsoft in the development of all versions of Microsoft Windows since Microsoft Windows 2003, is available at https://msdn.microsoft.com/en-au/library/ms995349.aspx.

Further information on secure coding practices is available at https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21274.

Web application development

Protecting web applications

Even when a web application only contains public information, there remains a need to protect the integrity and availability of the information processed by the web application and the system it is hosted on.

Web application frameworks

Web application frameworks can be leveraged by software developers to enhance the security of a web application while decreasing development time. These resources can assist software developers to securely implement complex components such as session management, input handling and cryptographic operations.

Security Control: 1239; Revision: 3; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Robust web application frameworks are used to aid in the development of secure web applications.

Input handling

Most web application vulnerabilities are caused by the lack of secure input handling. It is essential that web applications do not trust any input such as the website address and its parameters, Hypertext Markup Language (HTML) form data, cookie values and Hypertext Transfer Protocol (HTTP) request headers without validating or sanitising it. Examples of validation and sanitisation include:

- ensuring a telephone form field contains only numerals
- ensuring data used in a Structured Query Language (SQL) query is sanitised properly
- ensuring Unicode input is handled appropriately.

Security Control: 1240; Revision: 2; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Must
Validation and/or sanitisation is performed on all input handled by a web application.

Output encoding

The likelihood of cross-site scripting and other content injection attacks can be reduced through the use of contextual output encoding. The most common example of output encoding is the use of HTML entities. Performing HTML entity

encoding causes potentially dangerous HTML characters such as '<', '>' and '&' to be converted into their encoded equivalents '<', '>' and '&'.

Output encoding is particularly useful where external data sources, which may not be subject to the same level of input filtering, are output to users.

Security Control: 1241; Revision: 3; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Must
Output encoding is performed on all output produced by a web application.

Web browser-based security controls

Web browser-based security controls such as Content Security Policy, HTTP Strict Transport Security (HSTS) and Frame Options can be leveraged by web applications to help protect both web applications and their users.

These security controls are implemented by the web application via the insertion of HTTP headers containing security policy in outgoing responses. Web browsers then apply the security controls according to the defined policy. Since the security controls are applied via HTTP headers, it makes it possible to apply the security controls to legacy or proprietary web applications where changes to the source code are impractical.

Security Control: 1424; Revision: 2; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
Web browser-based security controls are implemented for web applications in order to help protect both web applications and their users.

Open Web Application Security Project

The Open Web Application Security Project (OWASP) provides a comprehensive resource to consult when developing web applications.

Security Control: 0971; Revision: 6; Updated: Sep-18; Applicability: O, P, S, TS; Priority: Should
*The OWASP **Development Guide** is followed when developing web applications.*

Further information

Further information on auditing of web applications can be found in the ***Event logging and auditing*** section of the ***Guidelines for system monitoring***.

Further information on implementing web browser-based security controls can be found in the Australian Cyber Security Centre (ACSC)'s ***Protecting Web Applications and Users*** publication at https://www.acsc.gov.au/publications/protect/Protecting_Web_Apps.pdf.

Further information on web application security is available at https://www.owasp.org/. The OWASP ***Development Guide*** can be obtained from https://github.com/OWASP/DevGuide.

Further information on common web application frameworks for different programming languages, including a comparison of their functionality, is available at https://en.wikipedia.org/wiki/Comparison_of_web_frameworks.