Columbia University
Web Application Security Standards and Practices

Objective and Scope

Effective Date: January 2011

This Web Application Security Standards and Practices document establishes a baseline of security related requirements for all Columbia University-supported web services and websites, including Columbia University-branded applications supported/hosted by 3rd parties.

This document is intended for personnel responsible for developing and supporting Columbia University's web applications, services, and websites. The purpose of this document is to provide coding standards, which are based on accepted industry practices, to minimize security exploits due to improper and nonstandard coding practices. It also provides references to information about common web security vulnerabilities to enhance understanding of the root causes of such issues and how to remediate them appropriately.

If you are responsible for developing and supporting Columbia University's web applications, services, and websites, you must adhere to the standards and practices established by this document.

1. Introduction

The materials presented in this document are obtained from the Open Web Application Security Project (OWASP), the SANS (SysAdmin, Audit, Network, Security) Institute, and other recognized sources of industry best practices.

OWASP is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security.

SANS Institute was established as a cooperative research and education organization. At the heart of SANS are the many security practitioners in varied global organizations from corporations to universities working together to help the entire information security community.

This document is divided into seven sections that cover the following topics:

Section	Description
Threat Risk Modeling	Brief description of approved threat risk modeling methodologies to provide context for the application of web security standards described in the next section.
Web Security Standards	Specifies coding standards and basic security practices that must be followed when developing and improving websites and web applications.
OWASP Application Security Checklist	A checklist of key items to review and verify effectiveness.
OWASP Top 10 Application Security Risks	Issues commonly identified as susceptible to exploitation using well-known techniques, and recommended remediation approaches.
SANS Top 25 Most Dangerous Software Errors	Commonly exploited coding mistakes and recommended remediation approaches.
Additional Security Best Practices	Supplemental security controls that may optionally be considered.
References	Hyperlinks to materials referenced within this document and suggestions for further reading.

You must read all sections and implement controls which are aligned with business and operational requirements.

If you are interested in using web application and website software scanning tools to scan your website to identify potential vulnerabilities and exploits, please contact the CUIT Security Office for assistance at security@columbia.edu. It is expected that you will take the necessary actions to remediate exposures revealed by the scans.

2. Threat Risk Modeling

Before considering the specific security features and controls described in this document, it is important to understand the context for the application of web security standards. Security features and controls should be implemented to remediate meaningful risks to a web application. Every system is different and you are the most knowledgeable about your own system and the risks it faces. This section provides a brief description of CUIT-approved threat risk modeling methodologies to assist you in first identifying and prioritizing the risks that should drive your subsequent selection of web security features and controls.

CUIT's recommended threat risk modeling methodology is the OWASP Threat Risk Model process:

http://www.owasp.org/index.php/Threat_Risk_Modeling

After you've performed the risk evaluation, you need to consider the controls to implement. To determine the type of security control that is needed, you should apply security control requirements using the Confidentiality, Integrity, Availability, and Accountability (CIAA) methodology as follows:

1) Determine whether a security control mechanism is required to ensure the Confidentiality, Integrity, Availability and/or Accountability of the data.

2) Using the CIAA approach, evaluate and rank the importance of each to prioritize what and where control mechanisms should be applied.

3. Web Security Standards

This section lists the web security standards which must be implemented by CU supported web applications, services, and sites.

Additionally, for web applications and websites that support e-commerce, you must read and comply with "Section H Additional Protections for Credit Card Information" in the University's "Registration and Protection of Systems Policy", which is described at:

http://policylibrary.columbia.edu/registration-and-protection-systems-policy

3.1 Deny access for exception conditions

Handling errors securely is critical in secure coding; especially exceptions that occur in the processing of a security control. It is important that these exceptions do not enable behavior that the established control would normally not allow. There are only three possible outcomes from a security mechanism:

1) allow the operation
2) disallow the operation
3) exception

In general, you must design your security mechanism so that a failure will follow the same execution path as disallowing the operation. For example, security methods like isAuthorized(), isAuthenticated(), and validate() should all return false if there is an exception during processing.

3.2 Validate all inputs and sanitize all outputs

All input data must be validated, input data validation should occur in the following sequence:

1) Decode the data before performing the validation – for example, check input strings to prevent the program from executing malicious commands, scripts, codes, etc.;
2) Check for length criteria - for example, determine if it is within the allowable predetermined minimum and maximum range;
3) Check for acceptable data types - for example, determine if it is a valid data type (e.g., characters or numbers only); and finally
4) Check for unacceptable data types – for example, determined whether data entered is non-characters, non-numeric, special characters.

For a few data validation examples, see the OWASP data validation webpage http://www.owasp.org/index.php/Data_Validation

All outputs must be sanitized to ensure outputs do not reveal too much information, especially for error messages which can provide too much information (e.g., default system generated messages) that an attacker can use to exploit security weaknesses. For error handling of input data, the error message should not reveal too much information.

For example, when there is an invalid user ID and/or password entered, the error message should not reveal what component entered, whether it's the user ID or password, which caused the error. The message should be general (e.g., invalid entry) and not reveal more information than necessary.

3.3 Maintain separation of duties

The concept of separation of duties is that the entity that approves an action, the entity that carries out an action, and the entity that monitors that action must be distinct. The goal is to eliminate the possibility of a single user from carrying out and concealing a prohibited action.

In general, application administrators should not be users of the application because application administrators inherit privileged access in the application. There are situations where the application administrator is also an application user. In such scenario, maintain separate accounts, one as the application administrator and one as the application user. The use of the application administrator account must be used exclusively for authorized administrative tasks only.

3.4 Verify all user authentication and authorization

There must be a security control mechanism to authenticate the identity of the user. This is typically handled with a User ID account and a password. The password must be sufficiently long and complex, consisting of alphanumeric characters and, if feasible, special characters. Authentication can be achieved using Columbia UNI via CAS. See the CAS website for more info http://cuit.columbia.edu/cas-authentication.

After the user is authenticated, a security control mechanism must also ensure that the user's access rights to the data must be limited to only his authorized access level. Implement user access with the least privilege required.

3.5 Assign user with the least privilege access level

Access to resources must be granted with the least privilege to ensure that accounts have the least amount of privilege required to perform their job function and responsibility. This encompasses user rights and resource permissions, including file and database permission. The user's access rights and privileges must be limited to perform only tasks that the user is authorized and not beyond his authority. Also, before the access is granted, ensure that appropriate authorization is obtained from the requestor's manager and the data owner or data steward (i.e., a person who has been given the authority by the data owner to approve data access on behalf of the owner).

For example, if the user only requires read access to the application, then this is all the permission that should be granted. Under no circumstances should the user be allowed

update privileges unless he has been explicitly authorized for both read and update access.

3.6 Establish secure default settings

Security related parameters settings, including passwords, must be secured and not user changeable. For example, by default, password length and complexity should be enabled and not be changed by the user. Also, some applications are supplied with one or more default user accounts and passwords. If these accounts are not used, then they should be disabled or removed. If the default accounts are needed, then the default passwords must be changed immediately to a new complex password.

If your application requires a default password either for the user to initially sign-on to the application or in the event of a password reset for forgotten password, then the default password must be a complex password and should be different for each user. The default password should also have an expiration period (e.g., not valid more than 24 hours) and onetime use only; thus, the system will force the user to change his password immediately after using the default password.

3.7 Keep the attack surface area to a minimum

The system attack surface is the collection of input points that the application has for an attacker. More points of entry into the application provide more avenues for an attacker to find a weakness. Every feature that is added to an application adds a certain amount of risk to the overall application. The aim for secure development is to reduce the overall risk by reducing the attack surface area. Each feature must function accordingly to application specification and business requirements and should not be allowed to perform functions beyond its intended purpose.

3.8 Keep security simple and avoid security by obscurity

Attack surface area and simplicity go hand in hand. Some programmers prefer overly complex approaches to what would otherwise be relatively straightforward and simple code. Do not add complexity when simplicity will achieve the same results. You must avoid the use of complex architectures when a simpler approach would be faster and more efficient.

Using obscurity to provide security control always fails when it is the only control. This is not to say that keeping secrets is a bad idea. It simply means that the security of key systems should not be reliant upon keeping details hidden. In other words, security plus obscurity is fine; obscurity by itself is not.

For example, the security of an application should not rely upon knowledge of the source code being kept secret. The security should rely upon many other factors, including

reasonable password policies and controls, defense in depth, transaction limits, network access controls, and audit trails.

3.9 Provide defense in depth

Defense in depth is a concept where one control would be reasonable, more controls that approach risks in different fashions are better. Controls, when used in depth, can make severe vulnerabilities very difficult to exploit and thus unlikely to occur. The more control layers, the more difficult it would be to circumvent them than a single control; but you should also be mindful that security controls should not be too complex making them difficult to manage and maintain.

With secure coding, this may take the form of combining controls from various elements such as: tier-based validation, centralized auditing controls, and requiring user activity to be logged.

3.10 Do not automatically trust access from other systems

Implicit trust of externally run systems is not warranted; therefore, system access validation must be checked when a request is initiated by the external system. All external systems should be treated in a similar fashion.

For example, when receiving requests from another system, the identity of the system must be validated and the permission checked before processing the requests. If inputs are received from the system, always validate the input before processing.

3.11 Maintain up-to-date security fixes and patches

Once security issues are identified, vendors will release security fixes or patches to prevent exploits of the vulnerability. Therefore, you must constantly review the security notifications and updates from the vendor and apply the security fixes and patches on a timely basis.

3.12 Maintain Audit Logs

You will never be able to prevent every attack. No matter how good your defenses, some things will slip through. It is critical that there be sufficient audit logs in place to be able to discover that an attack occurred.

Audit Logging procedures must be documented. Audit logs recording user activities, exceptions, and information security events must be produced.

Where technically feasible, recorded events must include: Security administration

activities; Restricted account access; Logon/logoff success and failure; Unsuccessful attempts to access systems or information; Dates and times of activity; Source of connection and access.

Logs are a target and therefore, they must be secured. Wherever possible, lock down log files so that only administrators have access to them. Ideally, spool logs to a separate log server. If someone compromises a computer, the first thing they go after is the logs.

Logs must be protected to prevent the following: alterations to the recorded messages; editing or deletion of log files; log storage capacity being exceeded; viewing of confidential information in logs by unauthorized individuals;

Audit logs retention must meet business requirements.

3.13 Supplemental Web Security Standard Requirements

- Ensure that Web Server backups are regularly performed.
- Configure a secure web content directory by configuring anti-spambot protection; if appropriate (e.g. CAPTCHAs') "No Follow" or other key word filtering.
- Ensure that a Web site requiring authenticated login deployment plan addresses the following:

  (a) The purpose(s) and function of the web server.

  (b) The information categories that will be stored, processed, and/or transmitted through the web server.

  (c) The Security Requirements for this information, as well as any additional services.

  (d) Retrieval of information from or stored on another host (e.g. back-end database, mail server, proxy server, etc.) along with the security requirements needed on the alternate host.

  (e) How information is to be published on the web server.

  (f) Control access to data on the web server.

- For more information about Recommended Guidelines for web server controls and configurations, please see http://cuit.columbia.edu/recommended-columbia-university-web-environment

4. OWASP Web Application Security Checklist

The 'OWASP Application Security Verification Standards' document provides a security checklist of keys controls for you to verify.

These are a few examples of control points to check:

- Verify that all pages and resources require authentication except those specifically intended to be public.
- Verify that all password fields do not echo the user's password when it is entered, and that password fields (or the forms that contain them) have autocomplete disabled.
- Verify that sessions are invalidated when the user logs out.
- Verify that users can only access URLs for which they possess specific authorization.

You must read and understand the 'OWASP Application Security Verification Standards - Web Application Standards' document. Each item on the checklist is associated with one or more verification level. The document describes the requirement for each verification level.

https://www.owasp.org/index.php/Projects/OWASP_Application_Security_Verification_Standard_Project

At a minimum, you must perform levels 1A and 1B verifications to ensure appropriate security measures are implemented.

5. OWASP Top 10 Application Security Risks

The 'OWASP Top 10 Web Application Security Risks' and the 'OWASP Top 10 Mobile Risks' documents which OWASP has identified the ten most critical web application security and mobile risks and suggested remediation to implement.

These are a few examples of control points to check:

- 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.
- Cross-Site Scripting (XSS) flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping.

- Cross-Site Request Forgery (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.

- Insecure direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key.

You must read and understand the 'OWASP Top 10 Web Application Security Risks' and the 'OWASP Top 10 Mobile Risks' documents. These are the areas where majority of security exploits has been identified that can occur and therefore, preventive measures must be implemented.

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

You must implement the OWASP recommended remediation action items which counteract the exploits and vulnerabilities.

6. SANS Top 25 Most Dangerous Software Errors

The SANS Top 25 list is another list that identifies the most prevalent software errors that can result in exploits and vulnerabilities.

These are a few examples of control points to check:

- Buffer Copy without Checking Size of Input ('Classic Buffer Overflow').

- Reliance on Untrusted Inputs in a Security Decision.

- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal').

- Unrestricted Upload of File with Dangerous Type.

http://www.sans.org/top25-software-errors/

You must review and implement the SANS recommended remediation action items which counteract the exploits and vulnerabilities.

7. Additional Security Best Practices

The following are additional security best practices that you should consider implementing, if feasible.

- Provide a mechanism to ensure timeliness of data access. After the validation for data access is performed, it is expected that the data would be used immediately. However, if there is a time lapse between data access validation and data use, then a re-validation must be performed before granting access. For example, there should be a mechanism to monitor user inactivity time and require the user to re-validate after exceeding the allowable threshold. This can assure that the person has not walked away and an unauthorized person is using his account.

- Identify what permission level is actually required for database table access by the application and limit access accordingly. For example, if the application only requires read access, do not allow it update access. Implement least privilege required even for the application.

- DBA should access the database the same way as the application to ensure activity is logged and audited. DBA should access database via stored procedures to track their activity.

- Separate the web server from the application server (i.e., the web server should be physically or logically/virtually separate from the application server).

- Separate application from the database (i.e., the application server should be physically or logically/virtually separate from the database server).

- When performing a security evaluation process, involve all parties from technology, operational, and business areas with vested interest. Perform security process assessment by examining each component in details:

  1) Input control;
  2) Output control;
  3) Authentication control;
  4) Authorization control;
  5) Session management control;
  6) Logging and auditing; and
  7) Use of encryption.

- Keep security simple and configurable:

  - Plan security configuration from the beginning;
  - Separate users and administrator logins;
  - Practice least privilege – even in development;

- Configuration must be simple;
- Keep things secure by default;
- Configure logging levels;
- Clearly document all security configurations.

- Document program source codes and ensure that documentation is maintained and kept up-to-date.

- Review log activity and audit the logs for exceptions.

8. References

More information is available in the OWASP websites:

Secure Coding Principles

http://www.owasp.org/index.php/Secure_Coding_Principles

Application Security FAQ website

http://www.owasp.org/index.php/OWASP_AppSec_FAQ

Version Tracking History

Change history comments	Version	Date
Initial Web Security Standards and Practices document created by Larry Lee, Joel Rosenblatt, Andrew Johnston, Carol Kassel, Josh Freeman, and IANS.	V1	1/11
Document updated by Larry Lee to include reference to OWASP Top 10 Mobile Risks. Document reviewed by Joel Rosenblatt, Marty Wren, Andrew Johnston, Seth Theriault, and Joe Rini.	V1.1	12/12
Document updated by Larry Lee to replace reference to the former E-Commerce: Electronic Protection of Credit Card Holder Information Policy with the Registration and Protection of Systems Policy in section 3.	V1.2	12/13
Document updated by Larry Lee to replace WIND reference with CAS in section 3.4 and also added section 3.13 Supplemental Web Security Requirements with additional requirements recommended by Internal Audit. Document reviewed by Elvira Spika, Jim Bossio, Joe Rini, Frank O'Donnell, Joel Rosenblatt, and David Balducci	V1.3	11/14