AltiusIT

Recommended Software Security Controls

Introduction
Altius IT's list of recommended software security controls is based upon security best practices. This list may not be complete and Altius IT recommends this list be augmented with additional controls. While not all risk can be eliminated, software security controls include preventive, detective, and corrective safeguards that help reduce risks to acceptable levels.

Software design
During the software design process certain decisions are made that later result in risks to the user, code, database, and data. By reviewing decisions made during your design and implementation process, you can identify areas may be a risk to the end solution.

Documentation
Database documentation should be made available to software developers. The documentation should include a description of all tables and fields, data length and expected values for a field, and any permissions assigned to the field.

Authentication
Since most applications rely on a user ID and password to grant access, applications should be designed so that an attacker cannot easily deduce the user ID.

User roles should be defined based on the principle of "least privilege". If a user role will not be modifying data, then the role should not be given the opportunity to edit, delete, or add data.

Programmers should ensure that the entire logon transaction including IDs and passwords is performed over a secure encrypted channel such as SSL.

Forms based authentication should use a POST request to ensure that the authentication credentials are not cached to browser history. The application should make use of "no cache" tags to help prevent someone from backing up to a login page and resubmitting a logon.

"Remember me" functionality should not be used as it can be leveraged by an attacker.

Ensure credentials and e-mail addresses are not hard coded in the application.

The user should be notified upon successful logon of:

- The date and time of the user's last logon
- The location of the user (as can best be determined) at last logon
- The number of unsuccessful logon attempts using this user ID since the last successful logon

AltiusIT

Recommended Software Security Controls

Passwords
Users play an important part in protecting information systems and data. By allowing non-complex passwords, user passwords are easy to crack and data will be vulnerable to attack. To be effective, require passwords to be complex with special characters.

Require users to change their passwords when they log in for the first time. In general, users should be required to change their passwords on a regular basis. The more critical the application, the more often the password should be changed. The application should require a user to enter their current password prior to accepting a new password. Users should not be allowed to re-use their ten (10) most recent passwords as they may have been compromised. When a user forgets a password, the password must be changed rather than “recovered".

Enforce a password rule that limits the number of password changes that a user may make in a given time period. It is important to ensure that users who request a password reset are authenticated. Recommended approaches include requiring the user to answer a unique set of questions for authentication. If possible, users should be prompted to answer different, randomly-selected questions every time they must authenticate. Users should be able to update their own personal question and answer profiles.

Passwords should never be stored in plain text, instead passwords should be hashed using an unique random salt per password.

Access controls
Users should not be able to access an unauthorized page by entering the URL address in a browser.

User activity should not be cached when handling sensitive information. If multiple individuals share a workstation, clicking the back arrow should not take a user to the URL of the last user’s login or pages visited.

Sessions should timeout after a period of inactivity (e.g. 15 minutes).

Controls should lock an account after a number of failed (e.g. 6) login attempts. Repeated failed attempts to authenticate a user should trigger a security incident, and lock out the user's account.

Use file system access rights only as a last resort when other options are not available.

Input controls

AltiusIT

Recommended Software Security Controls

Validate all input including parameters, arguments, cookies, anything read from the network, environment variables, request headers, URL components, files, database records and any external system that provides data to the application.

- Assume that all input is malicious. Validate all data entering the application against known good criteria. Use a common library of field validations to more efficiently and accurately confirm the integrity of the data. Validate all input for type, length, format (syntax), range, and business rules before accepting the data to be displayed, processed, or stored.
- Every field must have a list of acceptable values. For example, ensure that numeric values are within expected ranges that do not result in unanticipated consequences when used in calculations or control structures. If a dollar amount is entered, ensure that the user can’t input a negative number. By allowing negative numbers, the program might actually transfer money out of an account vs. depositing money into an account. The same for quantities, no negative numbers unless it is appropriate.
- Reject any input that does not strictly conform to specifications. Strip out or replace risky characters that can be used to attack the application.

Sanitize input using encode or escape data that modifies certain characters. Replace with hexadecimal sequence prior to exchanging with external components (e.g. database, LDAP server, web browser). Sanitize and remove the following characters from any user or dynamic database input (examples in VBScript):

- ' (escape the single quote) input = replace(input, "'", "''")
- " (double quote) input = replace(input, """", "")
-) (close parenthesis) input = replace(input, ")", "")
- ((open parenthesis) input = replace(input, "(", "")
- ; (semi-colon) input = replace(input, ";", "")
- - (dash) input = replace(input, "-", "")
- | (pipe) input = replace(input, "|", "")

Strip null characters from the end of a user supplied string. In general, any character that is not a number or letter (i.e. non-alphanumeric) should be URL encoded. Examples of such characters include the "!", "@", "#", "$", and "%" characters. Use HTML or URL encoding to wrap data and treat it as a literal. All input must be reduced to a pure format, the format that the server and database expects.

Verify the origin of form data through the use of unpredictable, unique nonces as hidden input form values. A nonce (number used once) is a value that can be uniquely identified and tracked as having been used. The application should also verify the origin of the nonce, that it was generated relatively recently, and that it was intended for the appropriate task. Typical process:

- User requests a page that can trigger actions, eg: create or deleting content.

- The requested page generates a nonce value and makes it part of the action (eg: a GET or POST parameter).
- User requests the action, software uses the nonce to validate the request.
- Upon successful nonce validation, the action is executed and the nonce is flagged as having been used.

Client-side input validation should not be used as a security defense. Reject “known bad” input but use layers of defense and do not rely only on this control as it assumes the developer knows everything that could possibly be malicious.

All headers, cookies, query strings, form fields, and hidden fields accepting input should be validated against acceptable data lists.

Encryption
Determine data that is critical or vulnerable and develop encryption schemes to protect the data from unauthorized users and use.

Pages that accept an ID and password should have this information encrypted both in transit and while stored.

Sensitive data should also be encrypted while in transit and stored.

Use currently accepted industry-standard cryptographic algorithms. Do not use self-signed SSL certificates, or default certificates. Ensure that SSL certificates and encryption settings are properly configured.

Keys, certificates, and passwords should be stored in a secure manner. The encryption solution used should support encryption key changes and maintenance procedures. Perform a code review to identify how keys, passwords, and other secrets are stored, loaded, processed, and cleared from memory.

Error handling
The proper handling of error messages plays an important role in secure software applications. In many instances, the server or database passes information back to the program and it is up to the program to interpret the message and take the appropriate action.

When an error occurs, the code should not continue normal execution that would result in incorrect information returned to the user or database.

Ensure that the application is built to gracefully handle all possible errors. Determine which errors should trigger a response to the end user. When such errors occur, the application should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Custom routines should handle errors without relying on server handling routines.

Controls should log unhandled errors to an event log that includes the day and time, user ID, error code, and, if possible, the line of code. This log file should be encrypted as it contains critical information.

Software failure
When an error occurs that causes the program (or a part of the program) to fail, the application should block the user from reaching the operating system, command line, or file system. The application should fail gracefully and securely without divulging details of the underlying code or infrastructure to the user. The action that caused the error should be logged to an encrypted file.

Ensure that the application has a “safe mode” to which it can return if something truly unexpected occurs. If all else fails, the application should log the user out and close the browser window.

Resource management
Store state information on the server side only or ensure client-side state variables have not been altered.

Multithreading solves problems with throughput and responsiveness but can introduce deadlocks and race conditions. Use thread-safe techniques to protect against race conditions that could harm system availability and/or data integrity. Consider the following guidelines when using multiple threads:

- Do not use ThreadAbort to terminate other threads. Calling Abort on another thread may cause problems without knowing what point that thread has reached in its processing.
- Do not use ThreadSuspend and ThreadResume to synchronize the activities of multiple threads. Instead, use Mutex, ManualResetEvent, AutoResetEvent, and Monitor.
- Do not control the execution of worker threads from your main program (e.g. using events). Instead, design the program so that worker threads are responsible for waiting until work is available, executing, and notifying other parts of your program when finished.
- Use caution when locking on instances, for example lock(this) in C# or SyncLock(Me) in Visual Basic. If other code in the application, external to the type, takes a lock on the object, deadlocks could occur.
- Do ensure that a thread that has entered a monitor always leaves that monitor, even if an exception occurs while the thread is in the monitor. The C# lock

statement and the Visual Basic SyncLock statement provide this behavior automatically, employing a finally block to ensure that MonitorExit is called. If you cannot ensure that Exit will be called, consider changing your design to use mutual exclusion (mutex). A mutex is automatically released when the thread that currently owns it terminates.

- Do use multiple threads for tasks that require different resources. Avoid assigning multiple threads to a single resource. For example, any task involving I/O benefits from having its own thread, because that thread will block during I/O operations and thus allow other threads to execute. User input is another resource that benefits from a dedicated thread. On a single-processor computer, a task that involves intensive computation coexists with user input and with tasks that involve I/O, but multiple computation-intensive tasks contend with each other.

Session management

Deploy software applications with secure default permissions.

Use sufficient randomness when generating session IDs or in other security-sensitive contexts.

Ensure security resources are properly configured to prevent access and/or modification by unintended individuals. Any client-side security checks should be verified and enforced on the server-side.

The user must be made aware of and agree with the use of cookie sessions. The content of the cookie must not contain or be used to obtain sensitive information. Users must be able to immediately delete cookies and the state associated with it. Any information stored within a cookie must not be disseminated to third parties without the user’s consent.

State mechanisms should not be used to authenticate users.

Session IDs should be unique to users, and issued after successful authentication. Session IDs should be randomly generated using a respected randomization source. The ID should never contain personal information. Session IDs should always be assigned and never chosen by the user. The keyspace of the token must as large as possible to combat guessing and other attacks.

Session IDs must be protected throughout their life cycle to prevent hijacking. They should have a time-out set for inactive sessions. Active sessions should have a set time to expire and regenerate a new session token. This reduces the time window that a hacker would have to break into a session. Session IDs should be protected with

SSL. Session IDs should change routinely and always during major transitions such as when moving to and from an SSL and when authenticating. For highly secure transactions re-authentication and a new session ID should be issued prior to processing the requested transaction. On log out, the session ID should be over-written.

Malicious code
All parameters must be examined for calls to external sources. Review the code for any instance where input from an HTTP request could be written into any of these external calls.

Build filters that verify that only expected data is included. If symbols are required, assure that they are converted to HTML.

Prepend and append a quote to all user input.

Give users access to only the required database stored procedures.

Avoid shell commands and system calls. In many cases there are language libraries that perform the same functions without using a system shell interpreter.

In the event of data that is not acceptable, there should be a mechanism in place to block and time out the session

Code that accepts input from users via an HTTP request must be reviewed to ensure that it can identify long input that could contribute to buffer overflow. Once inappropriate data is identified, the activity should be logged and the data dropped. All data input fields should have reasonable field lengths and specific data types. Limit the amount of text allowed in free form fields.

Logging
The software should make it easy to track a transaction without excessive effort or access to the system. Software should be:

- Auditable. All activities that affect user state or balances are formally tracked.
- Traceable. It's possible to determine where an activity occurs in all tiers of the application.
- High integrity. Logs cannot be overwritten or tampered by local or remote users.

Preserve a baseline of the network to be used as a comparison point in the event of a system failure. Copies of log files should be made at regular intervals depending on volume and size (daily, weekly, monthly, etc.). A common naming convention should be adopted with regards to logs, making them easier to index.

Log files should be copied to a protected area and retained for at least one year to assist in future investigations and monitoring. At least three months of history should be immediately available for analysis.

For sensitive information, logs should be written so that the log file attributes are such that only new information can be written (older records cannot be rewritten or deleted). If the application contains critical data, logs should also be written to a write once / read many (WORM) device. Log files are critical data and they should be encrypted.

Procedures should verify that logging is active and working properly:

- Ensure events are properly classified
- Review logging for performance delays
- Ensure compliance related logging cannot be "turned off"
- Verify access to log files is properly restricted
- To assist with investigations, systems should have their date and time synchronized using Network Time Protocol (NTP)

System events and activities that should be monitored and logged include:

- System administrator and system operator activities
- System start-ups and shut-downs
- Logging start-ups and shut-downs
- Backups
- Exceptions
- Security events
- Protection software
- Protection hardware (firewalls, routers, etc.)
- Intrusion Detection Systems (IDS) and Intrusion Prevention Systems
- Modifications to data characteristics including permissions, location, file type
- Authentication successes and failures (e.g. log in, log out, failed logins)

Application logging requires more than relying on server logs. Application logs help identify security incidents, establish baselines, providing information about problems and unusual conditions, assist with incident investigation, and help detect attacks. Application events and activities that should be monitored and logged include:

- Application authentication (e.g. successes, failures, logouts)
- Data audit trails (e.g. access to sensitive data, adding data, modifying data, deleting data, exporting and importing data)
- Input validation failures (e.g. protocol violations, unacceptable encodings, invalid parameter names and values)
- Output validation failures (e.g. database record mismatch, invalid data encoding)

- Suspicious behavior (e.g. multiple records deleted in a short period of time, invalid access attempts)
- Session management failures (e.g. cookie session identification value modification)
- Application errors and events (e.g. syntax and runtime errors, connectivity problems, third party service error messages, file system errors, sequencing failure)
- Higher-risk functionality (e.g. network connections, adding and deleting users, changes to privileges, assigning users to tokens, adding or deleting tokens, use of administrative privileges, access by application administrators, access to sensitive data, use of data encrypting keys, key changes, creating and deleting system-level objects, data import and export including screen-based reports, submission of user-generated content)
- Legal and other opt-ins (e.g. permissions to access credit history, terms of use, permission to receive marketing communications)
- Security events or warnings

Log entries and automated audit trails should include:

- Host name
- Date
- Time
- Application ID (e.g. name and version)
- Initiating process or origination of event (e.g. entry point URL, page, form)
- Code location (e.g. module, subroutine)
- User initiating action (e.g. user ID)
- Event type
- Result status (e.g. success, failure, defer)
- Resource (e.g. identity or name of affected data, component)
- Location (e.g. IP address or location)
- Severity of event (e.g. emergency, alert, fatal error, warning, information only)
- Other (e.g. parameters, debug information, system error message)

Testing

Software testing, when done correctly, can increase overall software quality of conformance by testing that the product conforms to its requirements. Testing includes, but is not limited to, unit and system testing, functional testing, performance testing, failover testing, usability testing, etc.

Thoroughly test the application to determine possible errors. Perform testing using both valid and invalid data. Determine in advance the appropriate programmatic response and compare with actual results obtained.

Testing of an application prior to moving it into a production environment should include a review of user role documentation and a review of the access control code.

Penetration testing should be performed to ensure that the application has been evaluated for security vulnerabilities.

Web Application and Server Configuration
Prevent against directory traversal and other risks by configuring the server to separate the operating system from the Web server.

Verify that assigned file and directory permissions are correctly applied using the principle of “least privilege”.

Disable any services that are not used by the Web server or application software.

Delete default accounts and change default passwords. Rename the default Administrator account or make it inaccessible. Delete all guest accounts.

Disable debugging functions. Ensure that debug code has been removed from software applications prior to moving the code into production.

Edit system generated error messages to provide as little information as possible to hackers.

Determine how the site will be administered. Document who has the rights to make changes, and when they can be made. Evaluate secure remote access solutions including Virtual Private Network (VPN), strong authentication with tokens, and certificates. Consider binding administrator functions to specific IP addresses using IP filtering.

Maintenance
Monitor sources for latest security vulnerabilities. Test and apply software patches in a timely manner. Critical patches and updates should be evaluated within 7 days of release and applied within 30 days.

Publication Information
Altius IT is a security audit, security consulting, and risk management firm. We are certified by the Information Systems Audit and Control Association (ISACA) as a Certified Information Systems Auditor (CISA), Certified in Risk and Information Systems Controls (CRISC), and Certified in the Governance of Enterprise Wide IT (CGEIT). For more information, please visit www.AltiusIT.com.