SSI Secure Code Checklist. Escuela de Ingenieria Informática of Oviedo (University of Oviedo)

Adapted from the OWASP Secure Coding Practices Quick Reference Guide v2.0

https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf

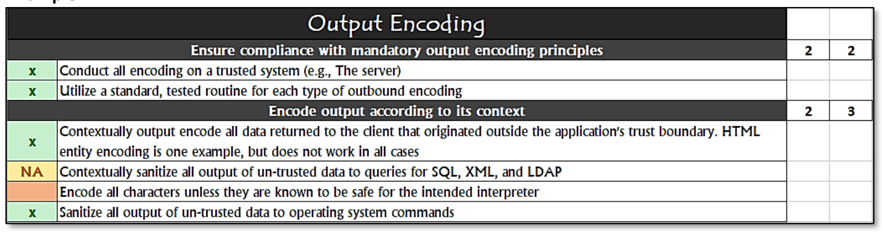
by José Manuel Redondo López

			Subcontrol Compliance %
Input / Output management			
1. Input Validation Checklist	**0**	**17**	0,00%
Ensure compliance with mandatory input validation principles	0	3	
Perform proper data encoding	0	4	
Implement secure management of data contents	0	4	
Validate data types and limits	0	3	
Perform data validation in a secure way	0	3	
2. Output Encoding	**0**	**6**	0,00%
Ensure compliance with mandatory output encoding principles	0	2	
Encode output according to its context	0	4	
Authentication and Authorization			
3. Access Control (Authorization)	**0**	**24**	0,00%
Ensure compliance with mandatory authorization principles	0	4	
Enforce proper restrictions only to authorized users	0	8	
Implement security best access control practices in bussiness logic code	0	5	
Implement proper authorization management	0	4	
Manage user accounts securely	0	3	
4. Authentication and Password Management	**0**	**35**	0,00%
Ensure compliance with mandatory authentication principles	0	6	
Implement proper password storage policies	0	3	
Enforce secure password transmission	0	2	
Enforce a secure password policy	0	6	
Implement secure authentication mechanism practices	0	5	
Manage account passwords securely in GUIs / application logic	0	11	
Implement a password log	0	2	
Cryptography in Applications			
5. Cryptographic Practices	**0**	**6**	0,00%
Ensure compliance with mandatory cryptography management policies	0	3	
Use only adequate cryptography modules	0	3	
6. Communication Security	**0**	**8**	0,00%
Ensure compliance with mandatory communication security principles	0	3	
Implement secure TLS management policies	0	5	
Programming Security			
7. General Coding Practices	**0**	**15**	0,00%
Ensure compliance with mandatory secure coding practices	0	1	
Implement a secure policy to use external APIs / Libraries	0	4	
Use secure programming techniques	0	7	
(EXTRA) counter-reverse engineering features	0	3	
8. Data Protection	**0**	**14**	0,00%
Implement secure data permission handling	0	4	
Encrypt data securely	0	1	
Implement secure information storage	0	2	
Remove any information the application gives about itself	0	5	
Implement secure GUI practices to deal with sensitive data	0	2	
9. Session Management	**0**	**19**	0,00%
Ensure compliance with mandatory session management practices	0	3	
Implement secure cookie management	0	4	
Implement secure login management	0	3	
Implement secure logout management	0	4	
Handle session data securely	0	5	
10. Error Handling and Logging	**0**	**23**	0,00%
Ensure compliance with mandatory logging practices	0	1	
Do not disclose excessive information on error messages	0	3	
Implement secure error handling	0	3	
Enable secure error log creation and management	0	8	
Log the adequate types of information related to security	0	8	
11. File Management	**0**	**14**	0,00%
Ensure compliance with mandatory file management practices	0	1	
Securely handle includes and references to files	0	5	
Implement secure file uploads, if this functionality is necessary	0	8	
12. Memory Management	**0**	**9**	0,00%
Ensure compliance with mandatory memory management principles	0	1	
Implement secure buffer/memory handling	0	4	
Implement secure policies to call functions	0	4	
External Systems			
13. System Configuration	**0**	**16**	0,00%
Use a secure software update policy	0	2	
Deploy a secure web server configuration regarding software elements	0	6	
Minimize and remove unnecessary files in the server	0	3	
Deploy a secure application configuration policy	0	5	
14. Database Security	**0**	**13**	0,00%
Ensure compliance with mandatory secure database management principles	0	1	
Securely handle types	0	2	
Enforce proper secure database permission	0	2	
Implement secure database management policies	0	8	
		Total Subcontrols	**14**

INSTRUCTIONS: Mark each control you think you comply with an "**x**" to recalculate compliance indexes on each of the 14 cathegories. If one of the controls does not apply to your application, simply put "**NA**" and it will be removed from the total amount of security controls that you need to comply on each cathegory. This page contains the global calculations, each individual Excel page contains the calculations of each cathegory (**14 pages total**)

Code Security Best Practices Evaluation Status		**Final Compliance**	**0,00%**
Code Security Group	**Implemented Subcontrols**	**Total Subcontrols**	**% Compliance**
Input / Output management	0	23	**0,00%**
Authentication and Authorization	0	59	**0,00%**
Cryptography in Applications	0	14	**0,00%**
Programming Security	0	94	**0,00%**
External Systems	0	29	**0,00%**
	0	**219**	

Example



Output Encoding			
Ensure compliance with mandatory output encoding principles		**2**	**2**
x	Conduct all encoding on a trusted system (e.g., The server)		
x	Utilize a standard, tested routine for each type of outbound encoding		
Encode output according to its context		**2**	**3**
x	Contextually output encode all data returned to the client that originated outside the application's trust boundary. HTML entity encoding is one example, but does not work in all cases		
NA	Contextually sanitize all output of un-trusted data to queries for SQL, XML, and LDAP		
	Encode all characters unless they are known to be safe for the intended interpreter		
x	Sanitize all output of un-trusted data to operating system commands		

1. Input Validation Checklist

	Ensure compliance with mandatory input validation principles	**0**	**3**
	Conduct all data validation on a trusted system (e.g., The server)		
	Identify all data sources and classify them into trusted and untrusted. Validate all data from untrusted sources(e.g., Databases, file streams, etc.)		
	There should be a centralized input validation routine for the application		
	Perform proper data encoding	**0**	**4**
	Determine if the system supports UTF-8 extended character sets and if so, validate after UTF-8 decoding is completed		
	Encode data to a common character set before validating (Canonicalize)		
	Specify proper character sets, such as UTF-8, for all sources of input		
	Verify that header values in both requests and responses contain only ASCII characters		
	Implement secure management of data contents	**0**	**4**
	If any potentially hazardous characters must be allowed as input,be sure that you implement additional controls like output encoding, secure task specific APIs and accounting for the utilization of that data throughout the application. Examples of common hazardous characters include: < > " ' % ( ) & + \\' \"		
	If your standard validation routine cannot address the following inputs, then they should be checked discretely		
	Check for null bytes (%00) Check for new line characters (%0d, %0a, \r, \n) Check for "dot-dot-slash" (../ or ..\) path alterations characters. In cases where UTF-8 extended character set encoding is supported, address alternate representation like: %c0%ae%c0%ae/ (Utilize canonicalization to address double encoding or other forms of obfuscation attacks)		
	Validate all input against a "white" list of allowed characters, whenever possible		
	Validate data types and limits	**0**	**3**
	Validate data length		
	Validate data range		
	Validate for expected data types		
	Perform data validation in a secure way	**0**	**3**
	All validation failures should result in input rejection		
	Validate all client provided data before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values). Be sure to include automated post backs from JavaScript, Flash or other embedded code		
	Validate data from redirects (An attacker may submit malicious content directly to the target of the redirect, thus circumventing application logic and any validation performed before the redirect)		

2. Output Encoding

	Ensure compliance with mandatory output encoding principles	**0**	**2**
	Conduct all encoding on a trusted system (e.g., The server)		
	Utilize a standard, tested routine for each type of outbound encoding		
	Encode output according to its context	**0**	**4**
	Contextually output encode all data returned to the client that originated outside the application's trust boundary. HTML entity encoding is one example, but does not work in all cases		
	Contextually sanitize all output of un-trusted data to queries for SQL, XML, and LDAP		
	Encode all characters unless they are known to be safe for the intended interpreter		
	Sanitize all output of un-trusted data to operating system commands		

3. Access Control (Authorization)

Ensure compliance with mandatory authorization principles		**0**	**4**
	Access controls should fail securely		
	Deny all access if the application cannot access its security configuration information		
	Use a single site-wide component to check access authorization. This includes libraries that call external authorization services		
	Use only trusted system objects, e.g. server-side session objects, for making access authorization decisions		
Enforce proper restrictions only to authorized users		**0**	**8**
	Restrict access to application data to only authorized users		
	Restrict access to files or other resources, including those outside the application's direct control, to only authorized users		
	Restrict access to protected functions to only authorized users		
	Restrict access to protected URLs to only authorized users		
	Restrict access to security-relevant configuration information to only authorized users		
	Restrict access to services to only authorized users		
	Restrict access to user and data attributes and policy information used by access controls		
	Restrict direct object references to only authorized users		
Implement security best access control practices in bussiness logic code		**0**	**5**
	Enforce application logic flows to comply with business rules		
	If state data must be stored on the client, use encryption and integrity checking on the server side to catch state tampering.		
	Limit the number of transactions a single user or device can perform in a given period of time. The transactions/time should be above the actual business requirement, but low enough to deter automated attacks		
	Server-side implementation and presentation layer representations of access control rules must match		
	Use the "referer" header as a supplemental check only, it should never be the sole authorization check, as it is can be spoofed		
Implement proper authorization management		**0**	**4**
	Create an Access Control Policy to document an application's business rules, data types and access authorization criteria and/or processes so that access can be properly provisioned and controlled. This includes identifying access requirements for both the data and system resources		
	Enforce authorization controls on every request, including those made by server-side scripts, "includes" and requests from rich client-side technologies like AJAX and Flash		
	If long authenticated sessions are allowed, periodically re-validate a user's authorization to ensure that their privileges have not changed and if they have, log the user out and force them to re-authenticate		
	Segregate privileged logic from other application code		
Manage user accounts securely		**0**	**3**
	Implement account auditing and enforce the disabling of unused accounts (e.g., After no more than 30 days from the expiration of an account's password.)		
	Service accounts or accounts supporting connections to or from external systems should have the least privilege possible		
	The application must support disabling of accounts and terminating sessions when authorization ceases (e.g., Changes to role, employment status, business process, etc.)		

4. Authentication and Password Management

	Ensure compliance with mandatory authentication principles	0	6
	All authentication controls must be enforced on a trusted system (e.g., The server)		
	All authentication controls should fail securely		
	Establish and utilize standard, tested, authentication services whenever possible		
	Require authentication for all pages and resources, except those specifically intended to be public		
	Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control		
	Use a centralized implementation for all authentication controls, including libraries that call external authentication services		

	Implement proper password storage policies	0	3
	Authentication credentials for accessing services external to the application should be encrypted and stored in a protected location on a trusted system (e.g., The server). The source code is NOT a secure location		
	If your application manages a credential store, it should ensure that only cryptographically strong one-way salted hashes of passwords are stored and that the table/file that stores the passwords and keys is writeable only by the application. (Do not use the MD5 algorithm if it can be avoided)		
	Password hashing must be implemented on a trusted system (e.g., The server).		

	Enforce secure password transmission	0	2
	Only send non-temporary passwords over an encrypted connection or as encrypted data, such as in an encrypted email. Temporary passwords associated with email resets may be an exception		
	Use only HTTP POST requests to transmit authentication credentials		

	Enforce a secure password policy	0	6
	Enforce password complexity requirements established by policy or regulation. Authentication credentials should be sufficient to withstand attacks that are typical of the threats in the deployed environment. (e.g., requiring the use of alphabetic as well as numeric and/or special characters)		
	Change all vendor-supplied default passwords and user IDs or disable the associated accounts		
	Enforce password changes based on requirements established in policy or regulation. Critical systems may require more frequent changes. The time between resets must be administratively controlled		
	Enforce password length requirements established by policy or regulation. Eight characters is commonly used, but 16 is better or consider the use of multi-word pass phrases		
	Passwords should be at least one day old before they can be changed, to prevent attacks on password re-use		
	Prevent password re-use		

	Implement secure authentication mechanism practices	0	5
	Validate the authentication data only on completion of all data input, especially for sequential authentication implementations		
	Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code		
	If using third party code for authentication, inspect the code carefully to ensure it is not affected by any malicious code		
	Use Multi-Factor Authentication for highly sensitive or high value transactional accounts		
	Utilize authentication for connections to external systems that involve sensitive information or functions		

	Manage account passwords securely in GUIs / application logic	0	11
	All administrative and account management functions must beat least as secure as the primary authentication mechanism		
	Disable "remember me" functionality for password fields		
	Enforce account disabling after an established number of invalid login attempts (e.g. five attempts is common). The account must be disabled for a period of time sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed		
	Enforce the changing of temporary passwords on the next use		
	If using email based resets, only send email to a pre-registered address with a temporary link/password		
	Notify users when a password reset occurs		
	Password entry should be obscured on the user's screen.(e.g., on web forms use the input type "password")		
	Password reset and changing operations require the same level of controls as account creation and authentication		
	Password reset questions should support sufficiently random answers. (e.g., "favorite book" is a bad question because "The Bible" is a very common answer)		
	Re-authenticate users prior to performing critical operations		
	Temporary passwords and links should have a short expiration time		

	Implement a password log	0	2
	Implement monitoring to identify attacks against multiple user accounts, utilizing the same password. This attack pattern is used to bypass standard lockouts, when user IDs can be harvested or guessed		
	The last use (successful or unsuccessful) of a user account should be reported to the user at their next successful login		

5. Cryptographic Practices

Ensure compliance with mandatory cryptography management policies		**0**	**3**
	All cryptographic functions used to protect secrets from the application user must be implemented on a trusted system (e.g., The server)		
	Establish and utilize a policy and process for how cryptographic keys will be managed		
	Protect master secrets from unauthorized access		
Use only adequate cryptography modules		**0**	**3**
	All random numbers, random file names, random GUIDs, and random strings should be generated using the cryptographic module's approved random number generator when these random values are intended to be un-guessable		
	Cryptographic modules should fail securely		
	Cryptographic modules used by the application should be compliant to FIPS 140-2 or an equivalent standard. (See http://csrc.nist.gov/groups/STM/cmvp/validation.html)		

6. Communication Security

Ensure compliance with mandatory communication security principles		**0**	**3**
🟧	Filter parameters containing sensitive information from the HTTP referer, when linking to external sites		
🟧	Implement encryption for the transmission of all sensitive information. This should include TLS for protecting the connection and may be supplemented by discrete encryption of sensitive files or non-HTTP based connections		
🟧	Specify character encodings for all connections		
Implement secure TLS management policies		**0**	**5**
🟧	Failed TLS connections should not fall back to an insecure connection		
🟧	TLS certificates should be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required		
🟧	Utilize a single standard TLS implementation that is configured appropriately		
🟧	Utilize TLS connections for all content requiring authenticated access and for all other sensitive information		
🟧	Utilize TLS for connections to external systems that involve sensitive information or functions		

7. General Coding Practices

Ensure compliance with mandatory secure coding practices		**0**	**1**
	Use tested and approved managed code rather than creating new unmanaged code for common tasks		
Implement a secure policy to use external APIs / Libraries		**0**	**4**
🟧	Implement safe updating. If the application will utilize automatic updates, then use cryptographic signatures for your code and ensure your download clients verify those signatures. Use encrypted channels to transfer the code from the host server		
🟧	Review all secondary applications, third party code and libraries to determine business necessity and validate safe functionality, as these can introduce new vulnerabilities		
🟧	Use checksums or hashes to verify the integrity of interpreted code, libraries, executables, and configuration files		
🟧	Utilize task specific built-in APIs to conduct operating system tasks. Do not allow the application to issue commands directly to the Operating System, especially using application-initiated command shells		
Use secure programming techniques		**0**	**7**
🟧	Avoid calculation errors by understanding your programming language's underlying representation and how it interacts with numeric calculation. Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion, and casting between types, "not-a-number" calculations, and how your language handles numbers that are too large or too small for its underlying representation		
🟧	Do not pass user supplied data to any dynamic execution function		
🟧	Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage		
🟧	In cases where the application must run with elevated privileges, raise privileges as late as possible, and drop them as soon as possible		
🟧	Protect shared variables and resources from inappropriate concurrent access		
🟧	Restrict users from generating new code or altering existing code		
🟧	Utilize locking to prevent multiple simultaneous requests or use a synchronization mechanism to prevent race conditions		
(EXTRA) counter-reverse engineering features		**0**	**3**
🟧	Use a packer		
🟧	Use code obfuscation		
🟧	Use code trimming		

8. Data Protection

	Implement secure data permission handling	0	4
	Implement appropriate access controls for sensitive data stored on the server. This includes cached data, temporary files and data that should be accessible only by specific system users		
	Implement least privilege, restrict users to only the functionality, data and system information that is required to perform their tasks		
	Protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files as soon as they are no longer required.		
	Protect server-side source-code from being downloaded by a user		
	Encrypt data securely	0	1
	Encrypt highly sensitive stored information, like authentication verification data, even on the server side. Always use well vetted algorithms, see "Cryptographic Practices" for additional guidance		
	Implement secure information storage	0	2
	Do not store passwords, connection strings or other sensitive information in cleartext or in any non-cryptographically secure manner on the client side. This includes embedding in insecure formats like: MS viewstate, Adobe flash or compiled code		
	The application should support the removal of sensitive data when that data is no longer required. (e.g. personal information or certain financial data)		
	Remove any information the application gives about itself	0	5
	Do not include sensitive information in HTTP GET request parameters		
	Remove comments in user accessible production code that may reveal backend system or other sensitive information		
	(EXTRA) Remove file metadata and any kind of additional information files may give		
	(EXTRA) Do not use default templates of structures that may reveal technologies you are using		
	Remove unnecessary application and system documentation as this can reveal useful information to attackers		
	Implement secure GUI practices to deal with sensitive data	0	2
	Disable auto complete features on forms expected to contain sensitive information, including authentication		
	Disable client-side caching on pages containing sensitive information. Cache-Control: no-store, may be used in conjunction with the HTTP header control "Pragma: no-cache", which is less effective, but is HTTP/1.0 backward compatible		

9. Session Management

	Ensure compliance with mandatory session management practices	**0**	**3**
	Session identifier creation must always be done on a trusted system (e.g., The server)		
	Session management controls should use well vetted algorithms that ensure sufficiently random session identifiers		
	Use the server or framework's session management controls. The application should only recognize these session identifiers as valid		
	Implement secure cookie management	**0**	**4**
	Do not expose session identifiers in URLs, error messages or logs. Session identifiers should only be in the HTTP cookie header. For example, do not pass session identifiers as GET parameters		
	Set cookies with the HttpOnly attribute, unless you specifically require client-side scripts within your application to read or set a cookie's value		
	Set the "secure" attribute for cookies transmitted over an TLS connection		
	Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site		
	Implement secure login management	**0**	**3**
	Do not allow concurrent logins with the same user ID		
	Generate a new session identifier on any re-authentication		
	If a session was established before login, close that session and establish a new session after a successful login		
	Implement secure logout management	**0**	**4**
	Disallow persistent logins and enforce periodic session terminations, even when the session is active. Especially for applications supporting rich network connections or connecting to critical systems. Termination times should support business requirements and the user should receive sufficient notification to mitigate negative impacts		
	Establish a session inactivity timeout that is as short as possible, based on balancing risk and business functional requirements. In most cases it should be no more than several hours		
	Logout functionality should be available from all pages protected by authorization		
	Logout functionality should fully terminate the associated session or connection		
	Handle session data securely	**0**	**5**
	Generate a new session identifier and deactivate the old one periodically. (This can mitigate certain session hijacking scenarios where the original identifier was compromised)		
	Generate a new session identifier if the connection security changes from HTTP to HTTPS, as can occur during authentication. Within an application, it is recommended to consistently utilize HTTPS rather than switching between HTTP to HTTPS		
	Protect server-side session data from unauthorized access, by other users of the server, by implementing appropriate access controls on the server		
	Supplement standard session management for highly sensitive or critical operations by utilizing per-request, as opposed to per-session, strong random tokens or parameters		
	Supplement standard session management for sensitive server-side operations, like account management, by utilizing per-session strong random tokens or parameters. This method can be used to prevent Cross Site Request Forgery attacks		

10. Error Handling and Logging

Ensure compliance with mandatory logging practices		**0**	**1**
	All logging controls should be implemented on a trusted system (e.g., The server)		
Do not disclose excessive information on error messages		**0**	**3**
	Do not disclose sensitive information in error responses, including system details, session identifiers or account information		
	Implement generic error messages and use custom error pages		
	Use error handlers that do not display debugging or stack trace information		
Implement secure error handling		**0**	**3**
	Error handling logic associated with security controls should deny access by default		
	Properly free allocated memory when error conditions occur		
	The application should handle application errors and not rely on the server configuration		
Enable secure error log creation and management		**0**	**8**
	Do not store sensitive information in logs, including unnecessary system details, session identifiers or passwords		
	Ensure log entries that include un-trusted data will not execute as code in the intended log viewing interface or software		
	Ensure logs contain important log event data		
	Ensure that a mechanism exists to conduct log analysis		
	Logging controls should support both success and failure of specified security events		
	Restrict access to logs to only authorized individuals		
	Use a cryptographic hash function to validate log entry integrity		
	Utilize a master routine for all logging operations		
Log the adequate types of information related to security		**0**	**8**
	Log all access control failures		
	Log all administrative functions , including changes to the security configuration settings		
	Log all apparent tampering events, including unexpected changes to state data		
	Log all authentication attempts, especially failures		
	Log all backend TLS connection failures		
	Log all input validation failures		
	Log attempts to connect with invalid or expired session tokens▯og all system exceptions		
	Log cryptographic module failures		

11. File Management

	Ensure compliance with mandatory file management practices	**0**	**1**
	Ensure application files and resources are read-only		
	Securely handle includes and references to files	**0**	**5**
	Do not pass user supplied data directly to any dynamic include function		
	Do not pass directory or file paths, use index values mapped to pre-defined list of paths		
	Do not pass user supplied data into a dynamic redirect. If this must be allowed, then the redirect should accept only validated, relative path URLs		
	Never send the absolute file path to the client		
	When referencing existing files, use a whitelist of allowed file names and types. Validate the value of the parameter being passed and if it does not match one of the expected values, either reject it or use a hard-coded default file value for the content instead		
	Implement secure file uploads, if this functionality is necessary	**0**	**8**
	Do not save files in the same web context as the application. Files should either go to the content server or in the database		
	Implement safe uploading in UNIX by mounting the targeted file directory as a logical drive using the associated path or the chrooted environment		
	Limit the type of files that can be uploaded to only those types that are needed for business purposes		
	Prevent or restrict the uploading of any file that maybe interpreted by the web server		
	Require authentication before allowing a file to be uploaded		
	Scan user uploaded files for viruses and malware		
	Turn off execution privileges on file upload directories		
	Validate uploaded files are the expected type by checking file headers. Checking for file type by extension alone is not sufficient		

12. Memory Management

Ensure compliance with mandatory memory management principles		**0**	**1**
	Utilize input and output control for un-trusted data		
Implement secure buffer/memory handling		**0**	**4**
	Double check that the buffer is as large as specified		
	Properly free allocated memory upon the completion of functions and at all exit points		
	Specifically close resources, do not rely on garbage collection.(e.g., connection objects, file handles, etc.)		
	Use non-executable stacks when available		
Implement secure policies to call functions		**0**	**4**
	Avoid the use of known vulnerable functions(e.g., printf, strcat, strcpy etc.)		
	Check buffer boundaries if calling the function in a loop and make sure there is no danger of writing past the allocated space		
	Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions		
	When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string		

13. System Configuration

	Use a secure software update policy	**0**	**2**
	Ensure servers, frameworks and system components are running the latest approved version		
	Ensure servers, frameworks and system components have all patches issued for the version in use		
	Deploy a secure web server configuration regarding software elements	**0**	**6**
	Define which HTTP methods, Get or Post, the application will support and whether it will be handled differently in different pages in the application		
	Disable unnecessary HTTP methods, such as WebDAV extensions. If an extended HTTP method that supports file handling is required, utilize a well-vetted authentication mechanism		
	If the webserver handles both HTTP 1.0 and 1.1, ensure that both are configured in a similar manor or insure that you understand any difference that may exist (e.g. handling of extended HTTP methods)		
	Remove unnecessary information from HTTP response headers related to the OS, web-server version and application frameworks		
	Restrict the web server, process and service accounts to the least privileges possible		
	Turn off directory listings		
	Minimize and remove unnecessary files in the server	**0**	**3**
	Prevent disclosure of your directory structure in the robots.txt file by placing directories not intended for public indexing into an isolated parent directory. Then "Disallow" that entire parent directory in the robots.txt file rather than Disallowing each individual directory		
	Remove all unnecessary functionality and files		
	Remove test code or any functionality not intended for production, prior to deployment		
	Deploy a secure application configuration policy	**0**	**5**
	Implement a software change control system to manage and record changes to the code both in development and production		
	Implement an asset management system and register system components and software in it		
	Isolate development environments from the production network and provide access only to authorized development and test groups. Development environments are often configured less securely than production environments and attackers may use this difference to discover shared weaknesses or as an avenue for exploitation		
	The security configuration store for the application should be able to be output in human readable form to support auditing		
	When exceptions occur, fail securely		

14. Database Security

Ensure compliance with mandatory secure database management principles		0	1
	Use strongly typed parameterized queries		
Securely handle types		0	2
	Ensure that variables are strongly typed		
	Utilize input validation and output encoding and be sure to address meta characters. If these fail, do not run the database command		
Enforce proper secure database permission		0	2
	The application should connect to the database with different credentials for every trust distinction (e.g., user, read-only user, guest, administrators		
	The application should use the lowest possible level of privilege when accessing the database		
Implement secure database management policies		0	8
	Close the connection as soon as possible		
	Connection strings should not be hard coded within the application. Connection strings should be stored in a separate configuration file on a trusted system and they should be encrypted.		
	Disable any default accounts that are not required to support business requirements		
	Remove or change all default database administrative passwords. Utilize strong passwords/phrases or implement multi-factor authentication		
	Remove unnecessary default vendor content(e.g., sample schemas)		
	Turn off all unnecessary database functionality (e.g., unnecessary stored procedures or services, utility packages, install only the minimum set of features and options required(surface area reduction))		
	Use secure credentials for database access		
	Use stored procedures to abstract data access and allow for the removal of permissions to the base tables in the database		