THIS IS NOT UP TO DATE

Cloudbase Security Checklist

The following chart is adapted from the Security Checklist provided by the Open Web Application Security Project (OWASP).

The OWASP Requirement column contains, verbatim, the secure coding practices that OWASP recommends. The Cloudbase Vulnerability column describes how and/or why the corresponding OWASP requirement is not met. If the column is left blank, then Cloudbase has met that requirement. If "n/a" appears in the column, then we believe the corresponding requirement is unrelated to or beyond the scope of Cloudbase. The Remediation Method column describes how Cloudbase can or has met the requirement. If left blank, the requirement is descriptive enough to describe the remediation method.
The Status column displays the status of the remediation method. The statuses are "Complete", "Incomplete", and "Incomplete. Low Priority". We plan to implement those that are incomplete. However, for those requirements of low priority, we have no plans to implement them due to time constraints

Input Validation

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Conduct all data validation on a trusted system (e.g., The server)		Cloudbase performs data validation on server-side PHP.	Complete
Identify all data sources and classify them into trusted and untrusted. Validate all data from untrusted sources (e.g., Databases, file streams, etc.)		Cloudbase validates all user input and all database queries.	Complete
There should be a centralized input validation routine for the application		Cloudbase uses a central routine that replaces dangerous characters with their escaped counterparts and otherwise sanitizes input.	Complete
Specify proper character sets, such as UTF-8, for all sources of input	Not yet implemented	Cloudbase will enforce proper character sets through PHP methods.	Inomplete

Encode data to a common character set before validating (*Canonicalize*)	Not yet implemented	Cloudbase will enforce proper character sets through PHP methods.	Incomplete
All validation failures should result in input rejection		Cloudbase rejects all input failures.	Complete
Determine if the system supports UTF-8 extended character sets and if so, validate after UTF-8 decoding is completed	Not yet implemented		Incomplete
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	Not yet implemented		Incomplete
Verify that header values in both requests and responses contain only ASCII characters	Not yet implemented		Incomplete
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)	Not yet implemented		Incomplete
Validate for expected data types	Not yet implemented	Cloudbase will validate data types	Incomplete
Validate data range			Complete
Validate data length	Not yet implemented	Cloudbase will validate data length	Incomplete

Validate all input against a "white" list of allowed characters, whenever possible		Cloudbase checks white-lists for appropriate characters if possible.	Complete
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: \ < > " ' % ( ) & + \ \' \"		All dangerous characters are replaced with escaped characters.	Complete
Check for null bytes (%00)			Complete
Check for new line characters (%0d, %0a, \r, \n)			Complete
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)	Not implemented	Cloudbase will not support extended character sets.	Incomplete

Output Encoding:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Conduct all encoding on a trusted system (e.g., The server)		All out put encoding is performed on the server.	Complete

Utilize a standard, tested routine for each type of outbound encoding			Complete
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			Complete
Encode all characters unless they are known to be safe for the intended interpreter	Not yet implemented		Incomplete
Contextually *sanitize* all output of un-trusted data to queries for SQL, XML, and LDAP			Complete
Sanitize all output of un-trusted data to operating system commands			Complete

Authentication and Password Management:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Require authentication for all pages and resources, except those specifically intended to be public		Cloudbase requires passwords to access all private pages.	Complete.
All authentication controls must be enforced on a trusted system (e.g., The server)		Authentication is performed on the Cloudbase server.	Complete
Establish and utilize standard, tested, authentication services whenever possible		A PHP hashing function and random number generator are used for storing passwords.	Complete
Use a centralized implementation for all authentication controls, including libraries that call external authentication services		Authentication control is isolated to an independent module.	
Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control		The authentication logic is performed by an isolated PHP routine on the server.	Complete
All authentication controls should fail securely			Complete
All administrative and account management functions must be at	Not yet implemented		Complete

least as secure as the primary authentication mechanism			
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 write-able only by the application. (Do not use the MD5 algorithm if it can be avoided)	The passwords are cryptographically stored. However, we are using the MD5 algorithm.	Cloudbase can switch to a more secure hashing algorithm such as ripemd128.	Incomplete.
Password hashing must be implemented on a trusted system (e.g., The server).		Hashing is perform only on the server.	Complete
Validate the authentication data only on completion of all data input, especially for *sequential authentication* implementations		Cloudbase only authenticates data when the user submits a complete form.	Complete
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		Authentication failure is conveyed by a single error message.	Complete

source code			
Utilize authentication for connections to external systems that involve sensitive information or functions	n/a		
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	n/a		
Use only HTTP POST requests to transmit authentication credentials		They system transmits credentials through POST.	Complete
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	Password resetting by e-mail is not yet implemented.	Cloudbase will only provide temporary passwords following a password reset request.	Incomplete
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	Not yet implemented.	Cloudbase will require passwords 8 characters in length that must have at least 1 alphanumeric and 1 numeric character.	Incomplete

of alphabetic as well as numeric and/or special characters)			
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	Not yet implemented.	Cloudbase will require passwords 8 characters in length that must have at least 1 alphanumeric and 1 numeric character.	Incomplete
Password entry should be obscured on the user's screen. (e.g., on web forms use the input type "password")			Complete
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	Not implemented.	We intend to prevent brute-force attacks at the firewall level.	Incomplete
Password reset and changing operations require the same level of controls as account creation and authentication.	Not yet implemented.	Users can change and reset their own passwords. Administrators can create users and reset their passwords.	Incomplete
Password reset questions should support sufficiently random answers. (e.g., "favorite	n/a		

book" is a bad question because “The Bible” is a very common answer)			
If using email based resets, only send email to a pre-registered address with a temporary link/password	Password resetting by e-mail is not yet implemented.	Cloudbase will only provide temporary passwords following a password reset request.	Incomplete
Temporary passwords and links should have a short expiration time	Password resetting by e-mail is not yet implemented.	Cloudbase will provide a short expiration time.	Incomplete
Enforce the changing of temporary passwords on the next use		We have no plans to implement this.	Incomplete. Low Priority.
Notify users when a password reset occurs	Not yet implemented.	Users will be notified.	Incomplete
Prevent password re-use		We have no plans to implement this.	Incomplete. Low Priority.
Passwords should be at least one day old before they can be changed, to prevent attacks on password re-use		We have no plans to implement this.	Incomplete. Low priority.
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		Password changes will not be enforced. We have no plans to implement this.	Incomplete. Low priority.

Disable "remember me" functionality for password fields		We have no plans to implement this.	Incomplete. Low priority.
The last use (successful or unsuccessful) of a user account should be reported to the user at their next successful login		We do not have a facility for tracking last user login success or failure. We do not plan to implement this.	Incomplete Low priority.
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		We do not plan to implement this.	Incomplete. Low priority.
Change all vendor-supplied default passwords and user IDs or disable the associated accounts		All vendor-supplied default accounts have been disabled.	Complete.
Re-authenticate users prior to performing critical operations		We have no plants to implement this.	Complete.
Use *Multi-Factor Authentication* for highly sensitive or high value transactional accounts	n/a		
If using third party code for authentication,	n/a		

inspect the code carefully to ensure it is not affected by any malicious code			

Session Management:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Use the server or framework's session management controls. The application should only recognize these session identifiers as valid		Cloudbase relies on local session management through PHP.	Complete.
Session identifier creation must always be done on a trusted system (e.g., The server)		Session identifiers are performed only on the server.	Complete.
Session management controls should use well vetted algorithms that ensure sufficiently random session identifiers		We assume PHP's algorithms are sufficient.	Complete.
Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site	Cloudbase does not address this issue.		Incomplete.

Logout functionality should fully terminate the associated session or connection		Logout terminates the session.	Complete
Logout functionality should be available from all pages protected by authorization		Logout is available on all private pages.	Complete.
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			Complete
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		Persistent logins will be prevented at the firewall level.	Incomplete.
If a session was established before login, close that session and establish a new session after a successful login		Logins create a new sessions.	Complete.

Generate a new session identifier on any re-authentication		There is a new session identifier for every login.	Complete.
Do not allow concurrent logins with the same user ID		We do not plan to implement this.	Incomplete. Low priority.
Do not expose session identifiers in URLs, error messages or logs. Session identifiers should only be located in the HTTP cookie header. For example, do not pass session identifiers as GET parameters		Session identifiers are never shared.	Complete.
Protect server side session data from unauthorized access, by other users of the server, by implementing appropriate access controls on the server	n/a		
Generate a new session identifier and deactivate the old one periodically. (This can mitigate certain session hijacking scenarios where the original identifier was compromised)		We do not plan to implement this.	Incomplete. Low priority.
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		Cloudbase does not switch from https to http during a session.	Complete.

consistently utilize HTTPS rather than switching between HTTP to HTTPS.			
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		We do not plan to implement this.	Incomplete. Low priority.
Supplement standard session management for highly sensitive or critical operations by utilizing per-request, as opposed to per-session, strong random tokens or parameters		We do not plan to implement this.	Incomplete. Low priority.
Set the "secure" attribute for cookies transmitted over an TLS connection	Not implemented		Incomplete
Set cookies with the HttpOnly attribute, unless you specifically require client-side scripts within your application to read or set a cookie's value	Not implemented		Incomplete

Access Control:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Use only trusted system objects, e.g. server side session objects, for making access authorization decisions			Complete.
Use a single site-wide component to check access authorization. This includes libraries that call external authorization services		Access is checked by a single PHP script. There are no external authorization services.	Complete.
Access controls should fail securely	Not yet implemented		Incomplete.
Deny all access if the application cannot access its security configuration information		Users can not login without providing credentials.	Complete.
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		Every request requires a credentials check.	Complete.
Segregate privileged logic from other application code		Privileged logic is separated from application code.	Complete.
Restrict access to files or other resources, including those outside the application's direct		Only system administrators will have access to resources beyond the web	Complete.

control, to only authorized users		application.	
Restrict access to protected URLs to only authorized users			Complete.
Restrict access to protected functions to only authorized users			Complete
Restrict direct object references to only authorized users			Complete
Restrict access to services to only authorized users			Complete
Restrict access to application data to only authorized users		Only administrators have access to the application data.	Complete.
Restrict access to user and data attributes and policy information used by access controls			Complete.
Restrict access security-relevant configuration information to only authorized users			Complete
Server side implementation and presentation layer representations of access control rules must match	Not yet implemented		Incomplete.

If *state data* must be stored on the client, use encryption and integrity checking on the server side to catch state tampering.	n/a		
Enforce application logic flows to comply with business rules		Of course.	Complete.
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		We have no plans to implement this. We will enforce a firewall to prevent repeated attack attempts.	Incomplete. Low priority.
Use the "referer" header as a supplemental check only, it should never be the sole authorization check, as it is can be spoofed		Referer header is not used as a check at all.	Complete.
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		We have no plans to implement this, but we will have session time-outs.	Incomplete. Low priority.
Implement account auditing and enforce the		We have no plans to implement this. The administrator is	Incomplete. Low priority.

disabling of unused accounts (e.g., After no more than 30 days from the expiration of an account’s password.)		responsible for disabling inactive accounts.	
The application must support disabling of accounts and terminating sessions when authorization ceases (e.g., Changes to role, employment status, business process, etc.)		We have no plans to implement this. This scenario is unlikely to occur with Cloudbase’s user base.	Incomplete. Low priority.
Service accounts or accounts supporting connections to or from external systems should have the least privilege possible	n/a		
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		See the SAD.	Complete.

Cryptographic Practices:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**

All cryptographic functions used to protect secrets from the application user must be implemented on a trusted system (e.g., The server)			Complete.
Protect master secrets from unauthorized access	n/a		
Cryptographic modules should fail securely			Complete.
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		PHP's random number generator is used.	Complete.
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)	The cryptographic module has not yet been updated from md5.		Incomplete
Establish and utilize a policy and process for how cryptographic keys will be managed	n/a	This will be the responsibility of the users.	

Error Handling and Logging:

OWASP Requirement	Cloudbase Vulnerability	Remediation Method	Status
Do not disclose sensitive information in error responses, including system details, session identifiers or account information			Complete.
Use error handlers that do not display debugging or stack trace information			Complete.
Implement generic error messages and use custom error pages			Complete.
The application should handle application errors and not rely on the server configuration			Complete.
Properly free allocated memory when error conditions occur			n/a
Error handling logic associated with security controls should deny access by default			Complete.
All logging controls should be implemented			Complete.

on a trusted system (e.g., The server)			
Logging controls should support both success and failure of specified security events	Cloudbase only supports failures.	Cloudbase will support successes.	Incomplete.
Ensure logs contain important *log event data*	Cloudbase does not support log event data as definited in this document.	We have no plans to implement log even data as defined.	Incomplete. Low Priority.
Ensure log entries that include un-trusted data will not execute as code in the intended log viewing interface or software			Complete.
Restrict access to logs to only authorized individuals		Only server administrators can access the logs.	Complete.
Utilize a master routine for all logging operations			Complete.
Do not store sensitive information in logs, including unnecessary system details, session identifiers or passwords			Complete.
Ensure that a mechanism exists to conduct log analysis		We do not have plans to implement this.	Incomplete. Low priority.
Log all input validation failures			Complete.

Log all authentication attempts, especially failures			Complete.
Log all access control failures			Complete.
Log all apparent tampering events, including unexpected changes to state data		We do not have plans to implement this.	Incomplete. Low priority.
Log attempts to connect with invalid or expired session tokens		We do not have plans to implement this.	Incomplete. Low priority.
Log all system exceptions		We're trying to do so.	Complete.
Log all administrative functions, including changes to the security configuration settings	Not yet implemented.	Cloudbase will log all system activities	Incomplete.
Log all backend TLS connection failures		We have no plans to implement this.	Incomplete. Low priority.
Log cryptographic module failures		We do not have plans to implement this.	Incomplete. Low priority.
Use a cryptographic hash function to validate log entry integrity		We do not have plans to implement this.	Incomplete. Low priority.

Data Protection:

OWASP Requirement	Cloudbase Vulnerability	Remediation Method	Status
Implement least privilege, restrict users to only the functionality, data and system information that is required to perform their tasks	Cloudbase plans to implement user access control.	Prevent normal users from accessing administrative tasks.	Incomplete
Protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files a soon as they are no longer required.	n/a	Cloudbase's data are not sensitive enough to warrant such protections.	
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	Cloudbase uses MD5 for hashing.	The system will be updated to use ripemd128 hashing.	Incomplete
Protect server-side source-code from being downloaded by a user		The code is open-source.	n/a
Do not store passwords, connection strings or other sensitive information in clear text or in any non-cryptographically secure manner on the client side. This		Cloudbase stores no sensitive information in clear text.	Complete

includes embedding in insecure formats like: MS viewstate, Adobe flash or compiled code			
Remove comments in user accessible production code that may reveal backend system or other sensitive information		Cloudbase stores no sensitive information in comments.	Complete
Remove unnecessary application and system documentation as this can reveal useful information to attackers		The system is open-source and will be well-documented.	n/a
Do not include sensitive information in HTTP GET request parameters			Complete
Disable auto complete features on forms expected to contain sensitive information, including authentication			Complete
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		Cloudbase stores no sensitive information.	n/a
The application should support the removal of		Cloudbase stores no such sensitive data.	n/a

sensitive data when that data is no longer required. (e.g. personal information or certain financial data)			
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		Only administrative users may access such data.	Comlete

Communication Security:
It is the responsibility of the client to manage his own security certificates. On our production server, we have implemented all these recommendations except for requiring secure connections.

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			n/a
TLS certificates should be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required			n/a

Failed TLS connections should not fall back to an insecure connection			n/a
Utilize TLS connections for all content requiring authenticated access and for all other sensitive information			n/a
Utilize TLS for connections to external systems that involve sensitive information or functions			n/a
Utilize a single standard TLS implementation that is configured appropriately			n/a
Specify character encodings for all connections			n/a
Filter parameters containing sensitive information from the HTTP referer, when linking to external sites			n/a

System Configuration:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Ensure servers, frameworks and system components are running the latest approved version			Complete.
Ensure servers, frameworks and system components have all patches issued for the version in use			Complete.
Turn off directory listings			Complete.
Restrict the web server, process and service accounts to the least privileges possible			Complete.
When exceptions occur, fail securely			Complete.
Remove all unnecessary functionality and files			Complete.
Remove test code or any functionality not intended for production, prior to deployment		We intend to remove all test code prior to deployment, but we have not yet reached that point.	Incomplete.
Prevent disclosure of your directory structure in the robots.txt file by placing directories not intended for public indexing into an isolated			Incomplete.

parent directory. Then "Disallow" that entire parent directory in the robots.txt file rather than Disallowing each individual directory			
Define which HTTP methods, Get or Post, the application will support and whether it will be handled differently in different pages in the application		The system will only use POST.	Complete.
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			Complete
If the web server 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)			Incomplete.
Remove unnecessary information from HTTP response headers related to the OS, web-server version and application frameworks			Incomplete

The security configuration store for the application should be able to be output in human readable form to support auditing		We do not plan to implement this.	Incomplete. Low priority.
Implement an asset management system and register system components and software in it		We have no plans to implement this.	Incomplete. Low priority.
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			Incomplete
Implement a software change control system to manage and record changes to the code both in development and production			Incomplete

Database Security:

OWASP Requirement	Cloudbase Vulnerability	Remediation Method	Status

Use strongly typed *parameterized queries*	Cloudbase does not use parameterized queries. Instead, it replaces dangerous characters with their escaped counterparts.	We do not have plans to implement parameterized queries.	Incomplete. Low priority.
Utilize input validation and output encoding and be sure to address meta characters. If these fail, do not run the database command			Complete.
Ensure that variables are strongly typed			n/a
The application should use the lowest possible level of privilege when accessing the database			Complete
Use secure credentials for database access			Complete
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.		The connection strings are stored separately and they are not stored on any version control system, but they are not encrypted.	Incomplete. Low priority.
Use stored procedures to abstract data access and allow for the removal of permissions to the base tables in the database	Not yet implemented		Incomplete.

Close the connection as soon as possible			Complete
Remove or change all default database administrative passwords. Utilize strong passwords/phrases or implement multi-factor authentication			Complete
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))		The database is access with the least necessary privileges.	Complete.
Remove unnecessary default vendor content (e.g., sample schemas)			Complete
Disable any default accounts that are not required to support business requirements			Complete
The application should connect to the database with different credentials for every trust distinction (e.g., user, read-only user, guest, administrators)		We do not have plans to implement this feature.	Incomplete. Low priority.

File Management:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Do not pass user supplied data directly to any dynamic include function			n/a
Require authentication before allowing a file to be uploaded			n/a
Limit the type of files that can be uploaded to only those types that are needed for business purposes			n/a
Validate uploaded files are the expected type by checking file headers. Checking for file type by extension alone is not sufficient			n/a
Do not save files in the same web context as the application. Files should either go to the content server or in the database.			n/a
Prevent or restrict the uploading of any file that may be interpreted by the web server.			n/a
Turn off execution privileges on file upload directories			n/a
Implement safe uploading in UNIX by			n/a

mounting the targeted file directory as a logical drive using the associated path or the chrooted environment			
When referencing existing files, use a white list 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			n/a
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			n/a
Do not pass directory or file paths, use index values mapped to pre-defined list of paths			n/a
Never send the absolute file path to the client			n/a
Ensure application files and resources are read-only			n/a
Scan user uploaded files for viruses and malware			n/a

Memory Management:

OWASP Requirement	Cloudbase Vulnerability	Remediation Method	Status
Utilize input and output control for un-trusted data			n/a
Double check that the buffer is as large as specified			n/a
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			n/a
Check buffer boundaries if calling the function in a loop and make sure there is no danger of writing past the allocated space			n/a
Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions			n/a
Specifically close resources, don't rely on garbage collection. (e.g.,			n/a

connection objects, file handles, etc.)			
Use non-executable stacks when available			n/a
Avoid the use of known vulnerable functions (e.g., printf, strcat, strcpy etc.)			n/a
Properly free allocated memory upon the completion of functions and at all exit points			n/a

General Coding Practices:

OWASP Requirement	**Cloudbase Vulnerability**	**Remediation Method**	**Status**
Use tested and approved managed code rather than creating new unmanaged code for common tasks		We try to use proven codes as much as possible.	Complete.
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 through the use of application initiated command shells		We avoid issuing any commands to the operating system as much as possible.	Complete.
Use checksums or hashes to verify the integrity of interpreted code, libraries, executables, and configuration files	We rely on the security that the underlying operating system provides.		Incomplete.
Utilize locking to prevent multiple simultaneous requests or use a synchronization mechanism to prevent race conditions		Cloudbase uses the client wins model to deal with concurrent access to the database.	Complete.
Protect shared variables and resources from inappropriate concurrent access		Cloudbase uses the client wins model to deal with concurrent access to the database.	Complete.
Explicitly initialize all your variables and other data stores, either during			Complete

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	n/a		
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		Cloudbase stores dollar values in cents to make use of integer precision. Cloudbase does store some aircraft-related values in floats, but the client says that this offer sufficient precision.	Complete
Do not pass user supplied data to any dynamic execution function			Complete.
Restrict users from generating new code or altering existing code			Complete
Review all secondary applications, third party code and libraries to		We do not have plans to validate any 3rd party code.	Incomplete. Low priority.

determine business necessity and validate safe functionality, as these can introduce new vulnerabilities			
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	We rely on the security of the operating system for all updates.		Incomplete.