CLOUD4C

01 January 2022

CODING SECURITY STANDARDS

Automation Team

CLOUD4C SERVICES PVT LTD

DOCUMENT CONTROL:

Preparation

Draft	Author	Date
1.0	M.Venkataniranjan	8/01/2017
2.0	M.Venkataniranjan	11/07/2017
2.1	M.Venkataniranjan	06/01/2018
2.2	M.Venkataniranjan	10/10/2018
2.2	M.Venkataniranjan	05/01/2019
2.2	M.Venkataniranjan	02/01/2020
2.2	M.Venkataniranjan	31/12/2020
2.2	Saikrishna.N	31/12/2021

Classification	Storage Location
Confidential	Shared Folder

Review & Approval

Reviewer	Version	Date	Reviewed Draft Version
R.S.Prasad Rao	1.0	16-01-2017	1.0
R.S.Prasad Rao	2.0	17-07-2017	2.0
R.S.Prasad Rao	2.1	07-01-2018	2.1
R.S.Prasad Rao	2.2	10-10-2018	2.2
R.S.Prasad Rao	2.2	07-01-2019	2.2
R.S.Prasad Rao	2.2	03-01-2020	2.2
R.S.Prasad Rao	2.2	01-01-2021	2.2
Jeeth (Automation Team-VP)	2.2	01-01-2022	2.2

Release

Release Version	Date Released
1.0	8/01/2017
2.0	17/07/2017
2.1	07/01/2018

2.2	10/10/2018
2.2	07/01/2019
2.2	03/01/2020
2.2	01/01/2021
2.2	01/01/2022

Distribution List

Name	**Designation**	**Department**
COE Engineers	COE Teams	Service delivery
All Bus	BU Heads & Employees	

Change Control

Version	**Change Reason**	**Effective Date**
2.0	Added new template	11/07/2017
2.1	Reviewed and updated version control	06/01/2018
2.2	Reviewed and, added Training section - 12, Review of the policy section -13	10/10/2018
2.2	Reviewed and no updates	07/01/2019
2.2	Reviewed and no updates	03/01/2020
2.2	Reviewed and no updates	31/12/2020
2.2	Reviewed and no updates	31/12/2021

STATEMENT OF CONFIDENTIALITY

This document contains proprietary trade secret and confidential information to be used solely for evaluating Cloud4C Services Private Ltd. The information contained herein is to be considered confidential. Customer, by receiving this document, agrees that neither this document nor the information disclosed herein, nor any part thereof, shall be reproduced or transferred to other documents, or used or disclosed to others for any purpose except as specifically authorized in writing by Cloud4C Services Private Ltd.

CONTENTS

1. PURPOSE & OBJECTIVES

PURPOSE

Coding Security standard seeks to ensure that applications developed by the Cloud4C Development Teams reflect secure coding practices. The recommendations below are provided as guidance for application software security requirements.

2. CHECKLIST

ID	Category	Compliance/ Process Detail
1.1	**Input Data Validation**	Always validate data in your PHP code. If you are using JavaScript to validate user input, there is always a chance that the user might have turned off JavaScript in her browser. In this case your app will not be able to validate the input. Validating in JavaScript is okay, but to guard against these types of problems then you should re-validate the data in PHP as well too.
		ð Input from $_GET, $_POST, $_COOKIE, and $_REQUEST is considered tainted.
		ð Understood that only some values in $_SERVER and $_ENV are untainted.
		ð $_SERVER['PHP_SELF'] is escaped where used.
		ð Input data is validated.
		ð 0 (null) is discarded in input.
		ð Length of input is bounded.
		ð Email addresses are validated.
		ð Application is aware of small, very large, zero, and negative numbers. Sci. notation too.
		ð Application checks for invisible, look-alike, and combining characters.
		ð Unicode control characters stripped out when required.
		ð Outputted data is sanitized.
		ð User-inputted HTML is sanitized with HTML Purifier.
		ð User-inputted CSS is sanitized using a white-list.
		ð Abusable properties (position, margin, etc.) are handled.
		ð CSS escape sequences are handled.
		ð JavaScript in CSS is discarded (expressions, behaviors, bindings).
		ð URLs are sanitized and unknown and unwanted protocols are disallowed.
		ð Embedded plugins are restricted from executing JS.
		ð Embedded plugin files (Flash movies) are embedded in a manner so that only the intended plugin is loaded.
		ð The application uses a safe encoding.
		o An encoding is specified using a HTTP header.
		o Inputted data is verified to be valid for your selected encoding if using an unsafe encoding.

ID	Category	Compliance/ Process Detail
1.2	**Guarding Against XSS Attacks**	Cross-site scripting attack (XSS attack) is an attack based on code injection into vulnerable web pages. The danger is a result of accepting unchecked input data and showing it in the browser. Suppose you have a comment form in your application that allows users to enter data, and on successful submission it shows all the comments. The user could possibly enter a comment that contains malicious JavaScript code in it. When the form is submitted, the data is sent to the server and stored into the database. Afterward, the comment is fetched from database and shown in the HTML page and the JavaScript code will run. The malicious JavaScript might redirect the user to a bad web page or a phishing website. To protect your application from these kinds of attacks, run the input data through strip_tags () to remove any tags present in it. When showing data in the browser, apply htmlentities()function on the data.
1.3	**Guarding Against XSS Attacks**	In a Cross Site Request Forgery (CSRF) attack, the attacker tricks the victim into loading sensitive information or making a transaction without their knowledge. This mainly occurs in web applications that are badly coded to trigger business logic using GET requests. The solution is to process any function that changes the database state in POST request, and avoid using $_REQUEST. Use $_GET to retrieve GET parameters, and use $_POST to retrieve POST parameters. In addition, there should be a random token called a CSRF token associated with each POST request. When the user logins into his/her account, the application should generate a random token and store it in the session. Whenever any form is displayed to the user, the token should be present in the page as a hidden input field. Application logic must check for the token and ensure that it matches the token present in the session
		ð CSRF issues are prevented with tokens/keys.
		o Referrers are not relied upon.
		o Pages that perform actions use POST.
		ð Important pages (logout, etc.) are protected.
		ð Your pages are not written in a way (i.e. JSON, JS-like) where they can be included and read on a remote website successfully.
		ð Aware that Flash can bypass referrer checks to load images and sound files.
		ð The following things will not reveal significant information if included remotely:
		ð Images.
		ð Pages that take a longer
		ð Existence or ordering of frames.
		ð Existence of a JS variable.
		ð Detected visit of a URL.
		ð Inclusion of your website in an inline frame with JS disabled does not reveal a threat.
		ð Application uses frame bursting code and sends the X-Frame-Options header.
1.4		

ID	Category	Compliance/ Process Detail

	SQL Injection Attacks	a look at the following example:
		`<?php`
		`$sql = "SELECT * FROM users WHERE name=:name and age=:age";`
		`$stmt = $db->prepare($sql);`
		`$stmt->execute(array(":name" => $name, ":age" => $age));`
		
		In the above code we provide the named parameters :name and :age to prepare(), which informs the database engine to pre-compile the query and attach the values to the named parameters later. When the call to execute() is made, the query is executed with the actual values of the named parameters. If you code this way, the attacker can’t inject malicious SQL as the query is already compiled and your database will be secure.
		
		Remote File Inclusion
		
		RFI allows an attacker to include remote file on the web server. The PHP language has an allow_url_fopen directive and if enabled it allows filesystem functions to use a URL which allows them to retrieve data from remote servers. An attacker can alter a variable that is passed to one of these functions to cause it to include malicious code hosted on his remote webserver.
		
		Local File Inclusion
		
		Local File Inclusion is almost similar to RFI, the only difference is that an attacker can include files which are hosted on the local server. The following is an example of local file inclusion vulnerability in PHP
		
		Command Injection
		
		Command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. It occurs when the user input is passed to shell via some commands like exec(), system() etc.
		
		The injection flaws can be prevented by following good practices such as:
		Validate all kind of content passed by the user to an application; this includes parameters passed as part of a GET or POST request to the application
		Limit the use of dynamic inputs from users to vulnerable functions either directly or via wrappers
		Set strong permissions on all well-known critical files on the webserver so that they cannot directly be accessed by an adversary
		Ensure that all variables are properly initialized prior to first use or disable register global, allow_url_fopen

ID	Category	Compliance/ Process Detail
1.5	**Protecting the File System**	As a developer you should always write your code in such a way that none of your operations put your file system at risk. Consider the following PHP that downloads a file according to a user supplied parameter: The script is very dangerous since it can serve files from any directory that is accessible to it, such as the session directory and system directories. The solution is to ensure the script does not try to access files from arbitrary directories.
1.6	**Protecting Session Data**	By default, session information is written to a temp directory. In the case of a shared hosting server, someone other than you can write a script and read session data easily. Therefore, you should not keep sensitive information like passwords or credit card numbers in a session. A good way to guard your session data is to encrypt the information stored in the session. This does not solve the problem completely since the encrypted data is not completely safe, but at least the information is not readable. You should also consider keeping your session data stored somewhere else, such as a database. PHP provides a method called session_set_save_handler() which can be used to persist data in session in your own way.
		ð Sessions only use cookies. (session.use_only_cookies)
		ð On logout, session data is destroyed.
		ð Session is recreated on authorization level change.
		ð Sites on the same server use different session storage dirs.

ID	Category	Compliance/ Process Detail
1.7	**Proper Error Handling**	It's good to know about all the errors that occur while we're developing an application, but when we make the application accessible to end users we should take care to hide the errors. If errors are shown to users, it may make our application vulnerable. So, the best approach is configuring your server differently for development and production environments. In production mode we need to turn off display_errors and display_start_up_errors settings. error_reporting and log_errors should be on so that we can log errors while hiding those from end users. Error messages can be put in server's error log instead of displaying them to a user with these configuration directives: log_errors = On display_errors = Off
1.8	**Guarding Included Files**	PHP scripts often include other PHP files that contain code for things like connecting to a database, etc. Some developers give the included files an extension like .inc. Files with this extension are not parsed by PHP by default if called directly and will be served as plain text to the users. If an attacker directly accesses the include file that contains database credentials, he now has access to all of your application's data. Always use the .php extension for included code files and keep them outside of directories directly accessible to users.
1.9	**Auto-loading classes**	PHP provides several ways to auto-load files containing classes that haven't yet been loaded. The older way is to use a magic global function called __autoload(). However you can only have one __autoload() function defined at once, so if you're including a library that also uses the __autoload() function, then you'll have a conflict.
2	**Maintain separation of duties**	The concept of separation of duties is that the entity that approves an action, the entity that carries out an action, and the entity that monitors that action must be distinct. The goal is to eliminate the possibility of a single user from carrying out and concealing a prohibited action. Separate the web server from the application server (i.e., the web server should be physically or logically/virtually separate from the application server). Separate application from the database (i.e., the application server should be physically or logically/virtually separate from the database server
2.1	**Authentication**	Verify all user authentication and authorization Is access to the source code of the Access Control System protected? There must be a security control mechanism to authenticate the identity of the user. This is typically handled with a User ID account and a password. The password must be sufficiently long and complex, consisting of alphanumeric characters and, if feasible, special characters.

ID	Category	Compliance/ Process Detail
2.2	**Authentication**	Assign user with the least privilege access level Access to resources must be granted with the least privilege to ensure that accounts have the least amount of privilege required to perform their job function and responsibility. This encompasses user rights and resource permissions, including file and database permission. The user's access rights and privileges must be limited to perform only tasks that the user is authorized and not beyond his authority. Identify what permission level is actually required for database table access by the application and limit access accordingly. For example, if the application only requires read access, do not allow it update access. Implement least privilege required even for the application.
		ð Strong passwords are used.
		ð Passwords stored safely.
		ð Bad password throttling.
		ð CAPTCHA is used.
		ð SSL used to prevent MITM.
		ð Passwords are not stored in a cookie.
		ð Passwords are hashed.
		ð Per-user salts are used.
		ð crypt () is used with sufficient number of rounds.
		ð MD5 is not used.
		ð Users are warned about obvious password recovery questions.
		ð Account recovery forms do not reveal email existence.
		ð Pages that send emails are throttled.
		ð A cryptographically secure PRNG is used for secret randomly-generated IDs (activation links, secret IDs, etc.).
		ð Suhosin is installed or you are not using rand() or mt_rand() for this.
		ð Anything that consumes a lot of resources should be throttled and limited.
		ð Pages that use 3rd-party APIs are throttled.
		ð You did not create your own encryption algorithm.
		ð Arguments to external programs (i.e. exec()) are validated.
		ð Generic internal and external redirect pages are secured.
		ð Precautions taken against the source code of your PHP pages being shown due to misconfiguration.
		ð Configuration and critical files are not in a web-accessible directory
2.3	**DataBases Access**	Provide a mechanism to ensure timeliness of data access. After the validation for data access is performed, it is expected that the data would be used immediately. However, if there is a time lapse between data access validation and data use, then a re-validation must be performed before granting access. For example, there should be a mechanism to monitor user inactivity time and require the user to re-validate after exceeding the allowable threshold. This can assure that the person has not walked away and an unauthorized person is using his account.
		ð Data inserted into the database is properly escaped or parameterized/prepared statements are used.

ID	Category	Compliance/ Process Detail
		ð addslashes() is not used.
		ð Application does not have more privileges to the database than necessary.
		ð Remote connections to the database are disabled if they are unnecessary.
2.4	**VA Test**	Conduct regular external and internal penetration tests to identify vulnerabilities and attack vectors that can be used to exploit enterprise systems successfully. Penetration testing should occur from outside the network perimeter (i.e., the Internet or wireless frequencies around an organization) as well as from within its boundaries (i.e., on the internal network) to simulate both outsider and insider attacks.
2.5	**PT Test**	Any user or system accounts used to perform penetration testing, should be controlled and monitored to make sure they are only being used for legitimate purposes, and are removed or restored to normal function after testing is over.
2.6	**Maintain up-to-date security fixes and patches**	Once security issues are identified, vendors will release security fixes or patches to prevent exploits of the vulner ability. Therefore, you must constantly review the security notifications and updates and apply the security fixes and patches on a timely basis.
2.7	**Maintain Audit Logs**	You will never be able to prevent every attack. No matter how good your defenses, some things will slip through. It is critical that there be sufficient audit logs in place to be able to discover that an attack occurred. Audit Logging procedures must be documented. Audit logs recording user activities, exceptions, and information security ty events must be produced. Review log activity and audit the logs for exceptions.
2.8	**Change control**	There must be formal change control process in place Team has to be made aware of Change Management
2.9	**Business Continuity**	Assess the impact on the portals, if certain services are not available for X hours There must be alternate arrangements in place in case of any disruptions to the service

3. TRAINING

Cloud4C shall ensure all team members undergo periodical training outlined in this policy.

4. REVIEW OF THE POLICY

- Weekly Meetings shall review suitability and adequacy Policy on Coding security standards document.
- This document will be reviewed and updated on an annual basis or when significant changes occur to the organization systems and information security standards.