**Security Code Review**

Before reviewing the source code in detail, ask the following questions to the team.

1. What kind of validations are there for user inputs? Are there length check validations? Are the inputs sanitized?

2. Is the code base clean? Is it written with OOP best practices? Is the application encapsulated?

3. Are errors and exceptions handled correctly? Does application log sensitive data? Does the application return stack trace to the client?

4. Which third party libraries are used? Are their APIs used correctly?

5. Does the application uses GET query strings? Does it pass sensitive data in query strings?)

6 .How is the application deployed to production? Are there any source code residing in production environment? Are config files are secured?

Once you have a basic understanding of the application architecture, you can start reviewing the code for security vulnerabilities. Here is a basic list to start from. This is not an exhausted list. Once you are done with this list, you can go to OWASP web site and search for more information.

* Make sure there are no application accounts, usernames and passwords in the source code. Developers might have entered these in source code and commented out during testing and forgot to delete it before checking in version control.
* Examine all the inputs that are not validated.
* Pay attention to error handling blocks and make sure they are handled properly. Verify that application does not return specific error messages such as giving insight into the database structure or invalid user ids vs password. Errors should be generic and not dispose inner workings of an application.
* When some exception occurs in application, generic error page should be displayed.
* Review the code for try-catch blocks that swallows exceptions. This is considered bad practice since the error cannot be tracked and the user can continue producing the error.
* Review code for resource leakage and memory, connection pooling and file handles. Make sure the application releases all these resources when it fails.
* Make sure User Id and Password validation handled correctly. Verify the application forces users to create strong passwords.
* Make sure User Id and Password is validated on the server side as soon as the http request is received.
* Verify that the passwords that are in database are encrypted and encryption algorithm that is used is strong.
* Review code for SQL Injection. Verify that parameterized queries are used instead of dynamic SQL statements.
* Review the stored procedures. They do not always protect against SQL injection.
* Verify that there is data validation for all external inputs.
* Make sure all inputs are sanitized against cross site scripting attacks.
* The application should not be using a black list of bad characters. This approached is not secured and require future maintenance.
* Examine logical conditional statements for authorization.
* Application should check authorization for every user request. Verify these logical conditional statements do not have loopholes.
* Verify that there are no stack traces displayed on UI.
* Verify application's thread safety implemented correctly.
* Verify that the session id that is created for authorized user is robust and cannot be predictable. Also, verify that the application creates and invalidates the sessions at the correct locations.
* Application should verify that the session is valid prior to servicing any user requests.
* Verify that the session ids are not passed through query string Http Get since any data on query string is logged in web server logs and also it is exposed to anybody who can see the URL.
* Review the code to see if the secure flag is set for cookies which prevents the cookie being transported over a non-secure channel.
* Verify that application has session timeout functionality and session settings are set correctly.
* Verify that the session is invalidated when user logs out.
* The application should identify the malicious activity and invalidate the user's session.
* Verify that the application does not have insecure cryptography algorithms.
* Verify that the server side code has input length validations.
* Verify that the application does not allow scripts getting inserted in log files.
* Search for code where the input is not validated and returned to the user in the format the user entered it.
* Verify that application does not trust the data in the database.
* Verify that there are no clear text passwords in the logs.
* Verify that application does not log user input directly. It should first validate the input then log it to prevent service denial attacks.
* Verify that the application logs user's malicious attacks including all requests that are made for authentication, authorization, session activities and data manipulation.
* Make sure there are no hard coded passwords or user ids in the source code.

Once you are done reviewing the source code, review the environment where the application is deployed.

* Make sure there is no source code residing.
* Make sure directories contain any files such as .bak, jars with different versions etc should be removed.