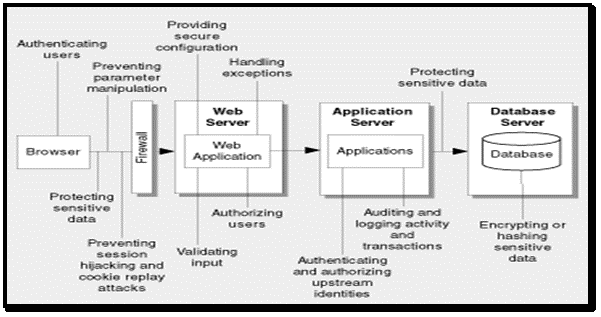
**Application architecture review**

Application architecture review can be defined as reviewing the current security controls in the application architecture. This helps a user to identify potential security flaws at an early stage and mitigate them before starting the development stage. Poor design of architecture may expose the application to many security loopholes. It is preferable to perform the architecture review at the design stage, as the cost and effort required for implementing security after development is high.

This document can be considered as the secure design guideline for the architects or as a checklist for a [penetration tester](https://www.infosecinstitute.com/courses/ethical-hacking-boot-camp/?utm_source=resources&utm_medium=infosec%20network&utm_campaign=course%20pricing&utm_content=hyperlink) to perform application architecture review as a part of the overall security assessment.

The following diagram shows some of the primary issues that must be addressed at the designing stage.



While doing the architecture review we can primarily focus on the following areas:

1. Application Architecture Documents
2. Deployment and Infrastructure Considerations
3. Input Validation
4. Authentication
5. Authorization
6. Configuration Management
7. Session Management
8. Cryptography
9. Parameter Manipulation
10. Exception Management
11. Auditing & Logging
12. Application Framework and Libraries

Additional category or points under any category can be added as per the requirement. Let’s have a look at each area separately:

## Application architecture documents:

The first thing to look for is the availability of the application architecture document. Every application should have a properly documented architecture diagram with a high-level explanation of the above points and a network connectivity diagram showing how different component are placed and secured.

## Deployment and infrastructure considerations:

Review the infrastructure on which the application is deployed. This can include reviewing the network, system, infrastructure performance monitoring, etc.

Some of the points which should be taken into consideration are as follows:

* **Components required for the application:** What is the OS supporting the application, hardware requirement, etc.?
* **Restrictions applied on the firewall:** Review the firewall policies defined for the application. What type of traffic is allowed and what type of traffic is blocked?
* **Port and Service requirement:** An application may communicate with other application as well. Identify which ports and services are required to be open for the application.
* **Component Segregation:** Components of the application should be segregated from each other. For example, the application server and database server should not reside in the same machine.
* **Disable clear text protocol:** Ports running clear text services like (HTTP, FTP, etc.) should be closed and not used for any part of the application.

## Input validation

Weak input validation is one of the main reasons for application security weakness. Proper input validation can help in preventing many attacks like Cross-Site Scripting, SQL Injection, etc. The validation should be applied on every input field (including the hidden form field) of all the pages. The best practice is to use a centralized approach.

Some of the points which should be taken into consideration are as follows:

* **Mechanism to validate the user inputs**: Check if the application is validating the user input or processing the input as it is.
* **Bypassing the validation**: Check how the user input is being validated. Is it possible to bypass the validation, for example, encoding the input? Identify if input validation depends on the application framework. Check whether there is any vulnerability in the framework through which a user can bypass the validation.
* **Centralized approach**: If custom implementation to validate the user input is present, check whether the approach is centralized.
* **Validating across all the tiers**: As a best practice, validation should be applied on all the layers, i.e. business layer, data layer, etc.
* **Addressing SQL Injection issue**: Input validation helps in mitigating SQL Injection issue to some extent. Check whether the application is safe against the SQL Injection vulnerability by using parameterized query in the back end.

## Authentication

Authentication is the act of verifying a user’s identity. In the application, it is achieved by providing the username and password. Weak authentication mechanism can result in bypassing the login process and accessing the application. This can lead to a major compromise. The [application should be designed with strong authentication](https://www.infosecinstitute.com/skills/courses/authentication-and-authorization/?utm_source=resources&utm_medium=infosec%20network&utm_campaign=skills%20pricing&utm_content=hyperlink).

Some of the points which should be taken into consideration are as follows:

* **Authentication control on the server side**: Make sure the credentials are verified at the server side and not on the client side. Client-side validation can be easily bypassed.
* **Secure channel for authentication**: Login credentials should always be sent through an encrypted channel. Credentials going through a clear text channel can be easily sniffed by the attacker.
* **Check whether the login page is served over HTTP protocol**. Check whether the application can be accessible on any other port where SSL certificate is not implemented.
* **Change password page**: Check whether the Old Password field is present in the Change Password page and verified as well.
* **Strong password policy**: An application should be configured to accept strong passwords only. Weak passwords can be brute forced easily.
* **Authentication cookie**: Check if SSL is implemented on an entire application and the authentication cookie is not sent in clear text on any page.
* **Service account**: Service account is an account under which a service application is running. A service account is required for the application and communicating with the database should have a restrictive set of privilege.
* **Default framework password**: Many application framework comes with a default password. Make sure the password is changed to a non-guessable strong password.

## Authorization

[Authorization](https://www.infosecinstitute.com/skills/courses/authentication-and-authorization/?utm_source=resources&utm_medium=infosec%20network&utm_campaign=skills%20pricing&utm_content=hyperlink) determines which resources can be accessed by the authenticated user. Weak authorization control can lead to Privileges Escalation attacks.

Some of the points which should be taken into consideration are as follows:

* **Privilege escalation and spoofing**: Privilege escalation happens when a user gets access to more resources than they are allowed or when the user can perform additional actions than they are allowed. Check the controls present when the user tries to escalate his/her privilege through manipulating the request or by directly accessing the unauthorized page/resource.
* **Direct object reference**: Check whether the application provides direct access to the objects based on the user-supplied input. This may allow the attacker to access the resources belonging to other users by bypassing the authorization. For example, downloading the invoices/statement of other users.

## Configuration management

Weak configuration should be avoided. Any sensitive information being stored in the configuration file can be extracted by the attacker.

Some of the points which should be taken into consideration are as follows:

* **Secure Hardening**: Make sure all the components required for the application are updated and the latest patches are applied on them. The default configuration should be changed wherever possible.
* **Sensitive Data**: Sensitive data like database connection string, encryption key, admin credentials or any other secret should not be stored as clear text in the code. Check whether the configuration file is secured against the unauthorized access.
* **Persistent cookie**: Storing sensitive data as plaintext in a persistent cookie should be avoided. The user can see and modify the clear text data. Check if the application is storing the clear text data in a persistent cookie.
* **Passing sensitive data using GET protocol**: The GET protocol sends the data in the query string. Sensitive information going in a GET request can be accessed from the browser history or logs.
* **Disable unused methods**: Verify that the application accepts only GET and POST methods. Other methods like TRACE, PUT, DELETE, etc. should be disabled.
* **Sensitive data over HTTP**: Communication between the components, such as between application server and database server, should be encrypted.

## Session management

The session is the track of the user activities. Strong session management plays an important role in the overall security of the application. Weakness in the session management may lead to serious attacks.

Some of the points which should be taken into consideration are as follows:

* **Use framework’s default session management**: Custom session management can have multiple vulnerabilities. Ensure there is no custom manager is being used and the application framework’s default session management is used.
* **Ensure session management best practices are followed:**
  + The session ID is random, long and unique.
  + The session is invalidating after logout.
  + The session ID is changing on successful authentication and re-authentication.
  + The session ID is not going in URL.
  + The session is timed out after a certain period of inactivity.
  + The session ID is going insecure channel like SSL.
  + Check whether cookie attributes (HttpOnly, Secure, path and domain) are secured.

## Cryptography

Applications frequently use [cryptography to secure the stored data](https://www.infosecinstitute.com/skills/learning-paths/introduction-to-applied-cryptography-and-cryptanalysis/?utm_source=resources&utm_medium=infosec%20network&utm_campaign=skills%20pricing&utm_content=hyperlink) or to protect the data in transit over an insecure channel.

Some of the points which should be taken into consideration are as follows:

* **Custom Implementation**: Designing a dedicated encryption mechanism may lead to weaker protection. Secure cryptographic service provided by the platform should be used. Check what type of encryption is being used in the application.
* **Encryption Key management**: Check whether there is any policy on encryption key management, i.e., Key generation, distribution, deletion, and expiration.
* **Securing encryption keys**: Encryption keys are used as an input to encrypt or decrypt the data. If encryption keys are compromised, the encrypted data can be decrypted and hence, will be no longer be secure.
* **Key recycle policy**: The key should be recycled after a certain period. Using the same key for a long time is not a safe practice.

## Parameter manipulation

With Parameter Manipulation attacks, the attacker modifies the data going from application to the web server. This can result in unauthorized access to services.

Some of the points which should be taken into consideration are as follows:

* **Validate all inputs from the client**: Validation applied on the client side may reduce the load on the server but relying only on client-side validation is not a safe practice. Client-side validation can be bypassed using a proxy tool. Check whether validation is applied on the server as well.
* **Do not rely on HTTP header**: Security decision in the application should not be based on the HTTP header. If the application is serving any page by checking only the “referrer” header, then an attacker can bypass this by changing the header in a proxy tool.
* **Encrypt the cookie**: Cookies can have data that is being used on the server to authorize the user. This type of data should be protected against unauthorized manipulation attacks.
* **Sensitive data in view state**: View state in an ASP.NET application can have sensitive data that is required to take the authorization decision on the server. Data in view state can be tempered if MAC (message authentication code) is not enabled. Check whether view state is protected using MAC.

## Exception management

Insecure exception handling can expose the valuable information, which can be used by the attacker to fine-tune his attack. Without exception management, information such as stack trace, framework details, server details, SQL query, internal path and other sensitive information can be exposed. Check whether centralized exception management is in place, with minimum information being displayed.

## Auditing & logging

Log files contain the record of the events. These events can be a success or failed login attempt, data retrieval, modification, deletion, network communication, etc. The logs should be monitored in real time.

Some of the points which should be taken into consideration are as follows:

* **Logging enabled**: Check if logging is enabled for application and platform level as well.
* **Log events**: Logs should be generated for all security-level important events like successful and failed authentication, data access, modification, network access, etc. The log should include time of the event, user identity, location with machine name, etc. Identify which events are being logged.
* **Logging sensitive data**: An application should not log sensitive data like user credentials, password hashes, credit card details, etc.
* **Storage, security, and analysis:**
  + The log file should be stored on a different partition than the one on which the application is running. The log file should be copied and moved to permanent storage for retention.
  + The log files must be protected against the unauthorized access, modification or deletion.
  + The log file should be analyzed on a regular interval.

## Application framework and libraries

Make sure that the application framework and libraries are up to date and relevant patches are applied on them. Verify that there is no default password being used in the framework (admin/admin, tomcat/tomcat, etc.). Check whether the old or vulnerable framework is in use.

## Conclusion

The above points represent the key areas for secure designing of the application. Implementing these points at the designing stage can reduce the overall cost and effort to secure the application. If the application is already deployed, secure architecture review is an important part of the overall security assessment and can help in fixing the existing vulnerabilities and improving the future design.