Module 1 Web Application security challenges

Introduction –

Security is one of the crucial aspects of quality of any software or any application. Security testing of web applications attempts to figure out various vulnerabilities, attacks, threats, viruses etc. related to the respective application. Security testing should attempt to consider as many as potential attacks as possible.

There are lot of challenges in the market to keep the application security program running successfully because, we have lot of dependencies and challenges as follows:

1. Available infrastructure is having loop holes.
2. Limitations for automated testing.
3. In depth defense is lacking.
4. Zero day attacks.

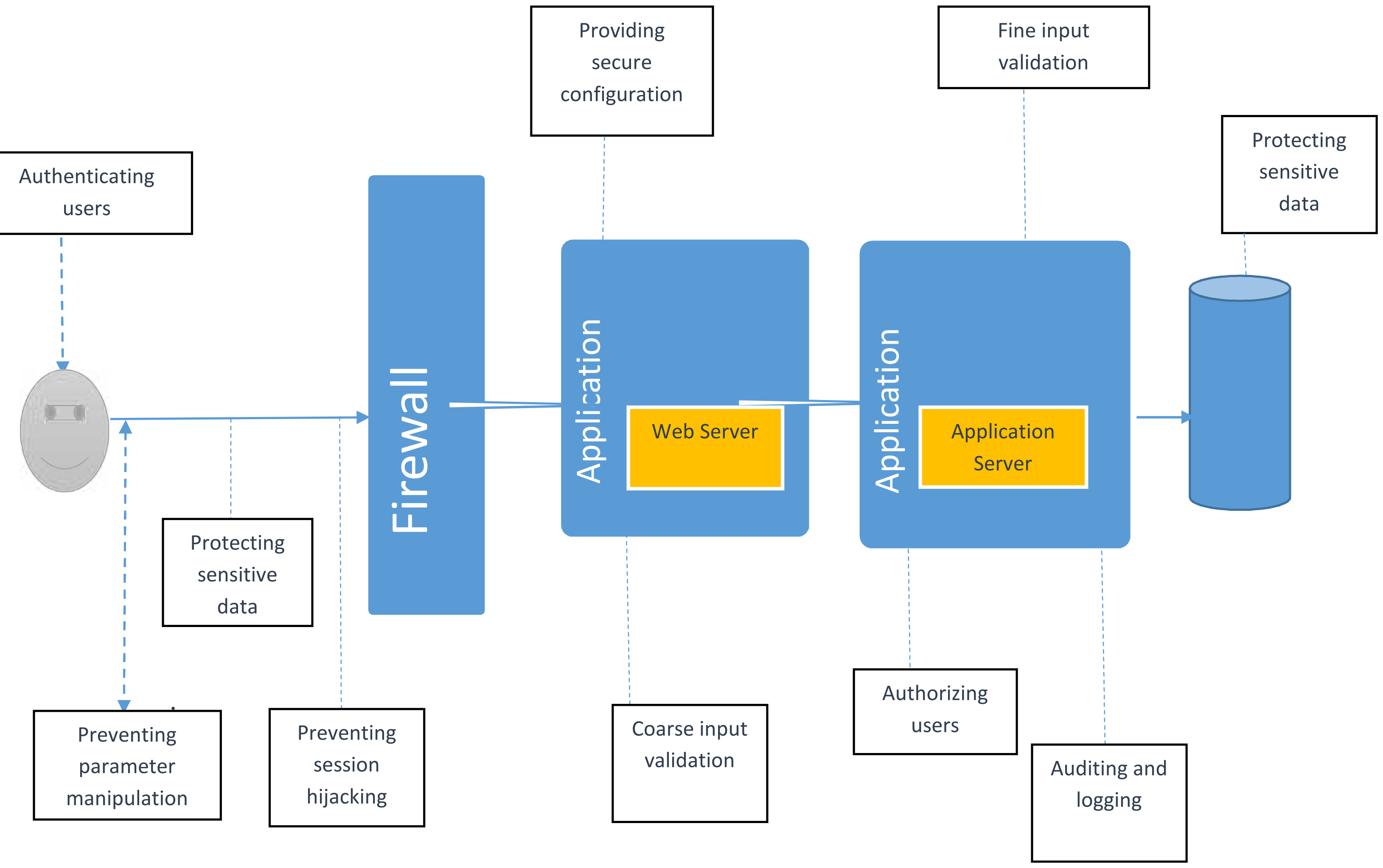
What makes Web applications vulnerable –

Without governance measures to manage security testing throughout the application delivery lifecycle, software teams can expose applications to several attacks as a result of:

- Analysts and architects viewing security as a network or IT issue, so that only a few organization security experts are aware of application-level threats.
- Teams expressing application security requirements as vague expectations or negative statements (e.g., you will not allow unprotected entry points that make test construction difficult).
- Testing application security late in the lifecycle — and only for hacking attempts.

Typical web application attacks –

The figure shows many points within a system that might require protection. Often, it is best to employ generic countermeasure concepts first to help ensure that you choose the technology best suited to your needs rather than one that claims to counter the latest hacking technique.


Providing secure configuration
Fine input validation
Authenticating users
Protecting sensitive data
Firewall
Application
Web Server
Application
Application Server
Protecting sensitive data
Preventing parameter manipulation
Preventing session hijacking
Coarse input validation
Authorizing users
Auditing and logging

Common types of Web application attacks

1. Impersonation

   Typing a different user’s credentials or changing a cookie or parameter to impersonate a user or pretend that the cookie originates from a different server.

   Common Cause –

   1. Using communications based authentication to allow access to any user’s data
   2. Using unencrypted credentials that can be captured and reused
   3. Storing credentials in cookies or parameters
   4. Using unproven authentication methods or authentication from the wrong trust domain
   5. Not permitting client software to authenticate the host

   Preventive Measures

   Use stringent authentication and protection for credential information using:

      1. Operating system (OS)- supplied frameworks
      2. Encrypted tokens such as session cookies
      3. Digital signatures

2. Tampering

   Changing or deleting a resource without authorization (e.g., defacing a Web site, altering data in transit)

   Common Cause –

   1. Trusting data sources without validation
   2. Sanitizing input to prevent execution of unwanted code
   3. Running with escalated privileges
   4. Leaving sensitive data unencrypted

   Preventive Measures

   1. Use OS security to lock down files, directories and other resources
   2. Validate your data
   3. Hash and sign data in transit (by using SSL or IPsec, for example)

3. Repudiation

   Attempting to destroy, hide or alter evidence that an action occurred (e.g., deleting logs, impersonating a user to request changes)

   Common Cause –

   1. Using a weak or missing authorization and authentication process
   2. Logging improperly
   3. Allowing sensitive information on unsecured communication channels

   Preventive Measures

   1. Use stringent authentication, transaction logs and digital signatures
   2. Audit

4. Information disclosure

   Revealing personally identifiable information (PII) such as passwords and credit card data, plus information about the application source and/or its host machines

   Common causes

   1. Allowing an authenticated user access to other users' data
   2. Allowing sensitive information on unsecured communication channels
   3. Selecting poor encryption algorithms and keys

   Preventive measures

   1. Store PII on a session (transitory) rather than permanent basis
   2. Use hashing and encryption for sensitive data whenever possible
   3. Match user data to user authentication

5. Denial of service (DoS)

   Flooding—sending many messages or simultaneous requests to overwhelm a server

   Lockout—sending a surge of requests to force a slow server response by consuming resources or causing the application to restart

   Common causes

1. Placing too many applications on a single server or placing conflicting applications on the same server
2. Neglecting to conduct comprehensive unit testing

Preventive measures

1. Filter packets using a firewall
2. Use a load balancer to control the number of requests from a single source
3. Use asynchronous protocols to handle processing-intensive requests and error recovery

6. Elevation of privilege

Exceeding normal access privileges to gain administrative rights or access to confidential files

Common causes

1. Running Web server processes as “root” or “administrator”
2. Using coding errors to allow buffer overflows and elevate application into a debug state •

Preventive measures

1. Use fewest-privileges context whenever possible
2. Use type-safe languages and compiler options to prevent or control buffer overflows

Basic guidelines for providing security for Web applications

By using security-specific processes to create applications, software development teams can guard against security violations. Specifically, you can apply several basic guidelines to existing applications and new or reengineered applications throughout your process to help achieve greater security and lower remediation costs, such as:

1. Discover and create baselines:

Conduct a complete inventory of applications and systems, including technical information (e.g., Internet Protocol [IP], Domain Name System [DNS], OS used), plus business information (e.g., who authorized the deployment? Who should be notified if the application fails?).

2. Assess and assign risks:

Rate applications and systems for risk—focusing on data stores, access control, user provisioning and rights management. Prioritize application vulnerabilities discovered during assessments.

3. Shield your application and control damage:

Stay on top of known security threats and apply available patches to your applications and/or infrastructure. If you cannot fix a security issue, use an application firewall, restrict access, disable the application or relocate it to minimize exposure.

4. Continuously monitor and review:

Schedule assessments as part of your documented change management process. When you close one out, immediately initiate a new discovery stage.